

零基础学习微信小程序开发，精选5个案例详细讲解，手把手带领读者快速入门小程序开发。
从开发思路、技术，到使用工具与案例，涉及小程序开发的方方面面。



李骏 边思 编著

微信小程序

开发入门及案例详解



机械工业出版社
China Machine Press

微信小程序：开发入门及案例详解

李骏 边思 编著

ISBN: 978-7-111-56210-8

本书纸版由机械工业出版社于2017年出版，电子版由华章分社（北京华章图文信息有限公司，北京奥维博世图书发行有限公司）全球范围内制作与发行。

版权所有，侵权必究

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @华章数媒

微信公众号 华章电子书（微信号：hzebook）

目录

对本书的赞誉

序一

序二

前言

第1章 初识小程序

1.1 简介

1.2 接入流程

1.2.1 注册小程序帐号

1.2.2 开发环境准备

1.3 第一个小程序

1.4 小结

第2章 小程序开发核心

2.1 简介

2.2 “徒手”创建小程序

2.3 框架主体文件

2.3.1 配置文件（app.json）

2.3.2 小程序逻辑（app.js）

2.3.3 全局样式（app.wxss）

2.4 框架页面文件

2.4.1 页面配置文件

2.4.2 页面逻辑文件 (JavaScript)

2.4.3 页面结构文件 (WXML)

2.4.4 页面样式文件 (WXSS)

2.5 模块化

2.5.1 模块化简介

2.5.2 文件作用域

2.5.3 模块的使用

2.5.4 其他

2.6 小结

第3章 布局

3.1 基本知识

3.1.1 盒子模型

3.1.2 块级元素

3.1.3 行内元素

3.1.4 行内块元素

3.2 浮动和定位

3.2.1 浮动

3.2.2 定位

3.3 Flex布局

3.3.1 基本概念

3.3.2 容器属性

3.3.3 项目属性

3.4 小结

第4章 组件

4.1 组件定义及属性

4.2 视图容器

4.2.1 view组件

4.2.2 scroll-view组件

4.2.3 滑块视图组件

4.3 基础组件

4.3.1 icon

4.3.2 text组件

4.3.3 progress组件

4.4 表单组件

4.4.1 radio组件

4.4.2 checkbox组件

4.4.3 switch组件

4.4.4 label组件

4.4.5 slider组件

4.4.6 picker组件

4.4.7 picker-view组件

4.4.8 input组件

4.4.9 textarea组件

4.4.10 button组件

4.4.11 form组件

4.5 导航组件

4.6 媒体组件

4.6.1 image

4.6.2 audio

4.6.3 video

4.7 地图组件

4.8 画布组件

4.9 客服会话

4.10 小结

第5章 API

5.1 网络

5.1.1 发起HTTPS请求

5.1.2 上传、下载

5.1.3 WebSocket

5.2 媒体

5.2.1 图片

5.2.2 录音

5.2.3 音频播放控制

5.2.4 音乐播放控制

5.2.5 音频组件控制

5.2.6 视频

5.2.7 视频组件控制

5.3 文件

5.4 数据缓存

5.4.1 保存数据

5.4.2 获取数据

5.4.3 获取本地数据信息

5.4.4 删除数据

5.4.5 清空数据

5.5 位置

5.5.1 获取位置

5.5.2 选择位置

5.5.3 查看位置

5.5.4 地图组件控制

5.6 设备

5.6.1 系统信息

5.6.2 网络状态

5.6.3 重力感应

5.6.4 罗盘

5.6.5 拨打电话

5.6.6 扫码

5.7 界面

5.7.1 交互反馈

5.7.2 设置导航条

5.7.3 导航

5.7.4 动画

5.7.5 绘图

5.7.6 下拉刷新

5.8 开放接口

5.8.1 登录

5.8.2 用户信息

5.8.3 微信支付

5.8.4 模板消息

5.8.5 客服消息

5.8.6 分享

5.8.7 获取二维码

5.9 小结

第6章 案例分析——豆瓣电影

6.1 准备工作

6.1.1 豆瓣API

6.1.2 跳转层

6.2 技术架构

6.3 公共模块开发

6.3.1 业务逻辑层

6.3.2 公共模块

6.4 页面构建

6.4.1 首页

6.4.2 详情页

6.5 页面逻辑开发

6.5.1 首页

6.5.2 详情页

6.6 小结

第7章 案例分析——驾考

7.1 业务流程

7.2 项目架构

7.2.1 功能点分析

7.2.2 项目结构图

7.2.3 数据接口

7.3 代码分析

7.3.1 小程序底层代码封装

7.3.2 首页

7.3.3 答题页

7.3.4 答题结果页

7.4 小结

第8章 案例分析——打赏

8.1 登录

8.1.1 登录流程

8.1.2 源码讲解

8.2 支付

8.3 小结

第9章 案例分析——日程表

9.1 业务流程

9.2 项目架构

9.2.1 功能点分析

9.2.2 项目结构图

9.3 代码分析

9.3.1 日程详情页

9.3.2 首页

9.3.3 日程管理页

9.4 小结

第10章 案例分析——多点商城

10.1 需求分析

10.2 技术架构

10.2.1 主界面架构

10.2.2 业务逻辑层

10.2.3 代理网络请求接口

10.2.4 本地模拟接口数据

10.2.5 widgets

10.2.6 全局样式控制

10.3 页面实现

10.3.1 主界面实现

10.3.2 首页与活动页

10.3.3 分类页与搜索页

10.3.4 支付流程

10.3.5 其他页面

10.4 小结

对本书的赞誉

近几年大前端技术飞速发展，2015年Facebook推出React Native以来，Native与HTML 5之间的界线已变得越来越模糊，而小程序的强势来袭更是推动这类技术在实际商业场景中的应用。本书将带领读者快速入门，掌握小程序开发技巧，一起加入小程序研发与探讨中。

——王楠，百度高级经理

小程序是非常棒的一款产品，它将跨平台研发技术与微信业务进行完美的融合，让开发者在微信环境中轻松开发出媲美原生体验的应用。大浪袭来时，卓立潮头的永远是认准风口的少数人。

——刘通，阿里技术专家

移动端和前端技术体系的融合已成为未来大趋势，小程序的横空出世无疑对这类技术在商业上的应用起到催化剂作用。本书在第一时间给出小程序实践思路，让初学者能快速入门，推荐大家仔细阅读，认真实践。

——王幸，阿里技术专家

小程序是可以轻松使用的跨平台技术，用炙手可热的前端技能开发出接近原生性能的应用。从此，你只需要专心构建功能或者服务，无需开发App，就可以令体验深入人心。通过作者用心写下的这本书，加入到连接人和微信的狂欢中来吧！

——陶铨，腾讯高级前端工程师

不输原生的用户体验，轻松的开发接入方式，更好的应用平台——与React Native相比，小程序借助于微信，相信将带来一场应用开发的变革。现在，这本书将带你参与到这场变革中，赶快加入吧！

——杨帆，多点总监

小程序是技术驱动业务的典范，它利用类似React Native、Weex等大前端渲染技术，完美解决了开发者业务接入的体验问题，这不禁让每一位技术从业者感到兴奋、自豪。本书第一时间给予最佳实践案例及学习思路，定能让读者受益匪浅。

——刘鹏，京东高级工程师

能在第一时间看到好友的书，感觉是如此的亲切，作为一个互联网老兵，能感到作者这么多年来在前端技术领域的坚持与执着。本书内容浅显易懂，有前端基础的同学能很快入门，希望能对小程序感兴趣的开发者带来一些帮助。

——李帅，U家工厂CEO

序一

2016年9月，微信刚一推出小程序，行业内便充满了微信对于App生态的各种猜想。媲美Native App上原生体验，基于微信强大流量入口和生态环境，比起之前推出的轻应用、直达号等，小程序的未来仍然充满了无限的想象力。小程序的发布，对App开发技术和前端H5技术的应用会产生重要的影响，尤其是对大多数创业公司而言，小程序会让研发成本更低，开发效率更高，产品迭代更快，流量获取更容易。因此随着小程序的不断成熟，对于未来业内产品形态、流量重构可能会掀起不小的波浪，是对新领域进行的大胆尝试。

本书作者李骏作为多点生活的资深前端架构师，曾就职于阿里、腾讯等知名互联网公司，具有顶尖的前端技术能力和丰富的实战经验，在第一时间便投入到微信小程序的实践中。本书可分为3部分，第一部分作为基础章节，介绍了第一个小程序的搭建流程，让大家能快速上手；同时对小程序框架原理进行了详细介绍，为后面学习组件、API打下基础。第二部分对小程序组件、API进行介绍，对组件、API的使用、注意事项进行详细讲解，并给出示例代码。最后一部分精选5个由浅入深的案例，对小程序研发进行实战讲解，涵盖了实际项目中可能涉及的技术方案和使用方法，具备很强的实战意义。在这本书

中，包含了作者在电商领域多年的前端经验总结和对当前主流架构的思考，希望读者们可以从中获取到自己想要的“干货”。

杨凯，多点生活技术VP

2017年1月于北京

序二

微信小程序如约而至，在继服务号、订阅号、企业号之后微信公众平台再一次做出了大胆尝试。相对于技术创新，产品色彩和利益驱动更浓厚。在当今互联网时代，信息流动越来越封闭，每个公司都想让信息、流量只进不出，尽量形成闭环，基于封闭的信息努力寻找一条盈利之路。基于这个逻辑、依托微信庞大的用户量和超强的用户粘性，微信将移动端跨平台技术与微信App进行深度集合，提高体验的同时提高开发效率。而由于小程序只能在微信中打开、分享，这使得小程序在技术层面和商业层面都形成良好的闭环，大大提升了微信的竞争力和抢占流量入口的能力。

小程序在本质上与React Native、Weex做的事大同小异，不同的是，小程序并不是将WXML完全原生化，现阶段仅仅是部分原生化，大部分渲染工作任由WebView完成，比如地图、textarea等组件的优化，可以说这种方式利用最小的投入显著提高了小程序的可用性。通过高度抽象的WXML和WXSS，可以从技术上通过限定一些Web技术子集，从而保障小程序的性能与体验，而之后小程序团队可在不影响开发者源码的情况下，随时通过升级Runtime与组件、API不断优化小程序性能与体验，甚至完全演变为全原生化渲染。这种设计是一种职

责上的划分，让开发者更关注业务，小程序团队则负责解决性能及底层问题，最快速地形成一套微信内的App生态。

小程序刚发布不久，本书作者从实践角度分析小程序研发技术并给出实践案例，让大家在最短时间内掌握小程序研发技术，帮助大家抢占入口中的入口。

韩笑跃，京东商城技术总监

2017年1月于北京

前言

自2016年9月21日微信小程序公布以来，微信技术群中关于小程序的讨论就没间断过，这是又一次创业的好机会，尤其是对中小企业扩大网络影响力很有利。我们在抓紧时间学习小程序的过程中，总结并实践了小程序的功能，并希望通过这本书传达给广大的读者。我们在编写过程中正临电商行业中最忙的几个月，双11、双12、圣诞节、元旦节等需求已经堆叠如山，我和边思白天处理公司需求，晚上编写书籍，几乎没有周末，这样坚持了几个月终于完成本书，直至交稿时才如释重负。

小程序刚发布不久，很多功能都还在不断更新中，本书内容在官方文档基础上进行补充说明，并给出实践案例，尝试给出现阶段尽可能完善的开发模式，适合小程序初学者入门。在小程序的学习过程中，我们做过很多尝试，这里我们仅仅提出了自己的一些实现方案和观点供大家参考。小程序整体推动还需要更多开发者参与，小程序还未正式公布前，便有很多公司及个人针对小程序研发过程中的痛点推出了各类三方框架，希望阅读本书后，大家也能提出自己的想法，积极参与小程序相关话题讨论，推动小程序研发方案优化与普及。

本书内容和组织结构

本书内容大致可分为三部分。

第一部分为基础部分，共3章，主要介绍小程序开发的基础知识。

第1章介绍微信小程序环境搭建、运行第一个小程序demo。

第2章介绍小程序核心知识，包括整体框架运行原理、规则，每个文件的作用，以及WXML与WXSS。

第3章介绍CSS布局基础，讲解了盒子模型、浮动、定位以及Flex布局。

第二部分为组件和API，共2章，主要讲解小程序组件、API相关知识。

第4章介绍小程序组件相关知识，主要内容包括视图容器、基础组件、表单组件、导航、媒体组件、地图、画布等。

第5章介绍小程序API相关知识，主要内容包括网络、媒体、文件、数据缓存、位置、设备、界面、开放接口等。

第三部分为案例实践，共5章，通过实际案例介绍如何开发小程序应用，包括一些思路和框架，以及部分代码和实现技巧。

第6章介绍如何开发豆瓣电影小程序，主要讲解一个最简单小程序的代码结构。

第7章介绍如何开发驾考小程序，主要讲解如何利用第三方API创建小程序引用。

第8章介绍如何开发打赏小程序，主要讲解小程序支付流程。

第9章介绍如何开发日程表小程序。

第10章介绍如何开发多点商城，主要讲解如何架构一个复杂小程序项目。

本书包含的所有案例代码可以到<https://github.com/wxapp-book> 下载。本书创作时间较短，如有疏漏，恳请各位读者斧正。

致谢

在这里感谢那些一直支持我们的人，感谢韩鑫、杨凯等公司领导对本书写作的支持，让我能空出时间投入到书籍编写，感谢吴怡编辑的辛勤工作，感谢龙伟湖对本书案例UI设计的友情支持，感谢杨帆、王庆平、许彬、张磊、范彩霞等同事在工作期间对我的各种支持，感谢罗东、杨小英等同事在这段时间为我分担工作，谢谢大家，正是因为你们的支持，才有了本书的面世。

最后特别感谢我的爱人张舒，一直相信我、支持我，一直为我默默付出，让我能全身心投入到工作中。

李骏

2016年12月

第1章 初识小程序

微信小程序自2016年9月21日内测以来，就引起广泛关注，越来越多的开发者开始研究如何使用它，在业界刮起了一阵不小的飓风。小程序不仅在商业上具备很大潜力，同时在技术上解决了一套代码多端运行和动态发版的两大痛点，用户在微信中扫一扫或搜一下即可打开具备原生体验的应用，这给开发者带来了很大的想象空间。小程序还在测试阶段就有大量开发者尝试为其开发各种框架，其中腾讯云还开发了一套微信小程序解决方案（<https://www.qcloud.com/solution/la.html>），由此可见小程序在业界的影响力非同小可。本章将介绍小程序的接入流程，并展示第一个小程序的开发过程，使读者快速入门，简单体验小程序的开发流程。

1.1 简介

按官方定义来讲，小程序是一种不需要下载安装即可使用的应用，它实现了应用“触手可及”的梦想，用户扫一扫或者搜一下即可打开应用。也体现了“用完即走”的理念，用户不用担心是否安装太多应用的问题。应用将无处不在，随时可用，但又无需安装卸载。

从技术角度来讲，小程序采用了类似React Native和Weex一样的解析技术，开发者可编写一套代码在多端运行（Android微信、iOS微信和浏览器容器），同时相比公众号H5应用，小程序具备更好的原生体验。严格来讲，小程序也是需要下载和安装的，只是由于技术方案以及官方规定小程序包容量不得超过1M，使得下载、安装（部署）过程特别快，用户在感官上察觉不到它在安装而已。为了达到用完即走、快速开发的目的，小程序提供了一套完整的开发框架、丰富的组件和API，相比React Native和Weex，小程序将技术与商业进行了完美的结合。

1.2 接入流程

小程序与订阅号、服务号、企业号是并行的体系，具有独立的注册、发布流程。开发小程序首先前需要在微信公众平台上注册小程序，完善基本信息，然后下载开发者工具进行编码，最后通过开发者工具提交代码，官方审核通过后便可发布。要注意的是，现阶段每个机构账号只允许注册最多50个小程序，每个小程序一年需要缴纳300元，所有小程序都需要绑定一个电子邮箱，一个手机号码最多只能绑定5个小程序。

1.2.1 注册小程序帐号

注册小程序帐号只需如下四步：

1) 在微信公众平台官网首页（mp.weixin.qq.com）点击右上角的“立即注册”按钮，如图1-1所示。



图1-1 注册小程序

2) 选择注册的帐号类型，选择“小程序”，如图1-2所示。

3) 进入帐号信息页面，填写未注册过公共平台、开放平台、企业号、未绑定个人号的邮箱，这个邮箱将作为以后登录小程序后台的帐号，如图1-3所示。

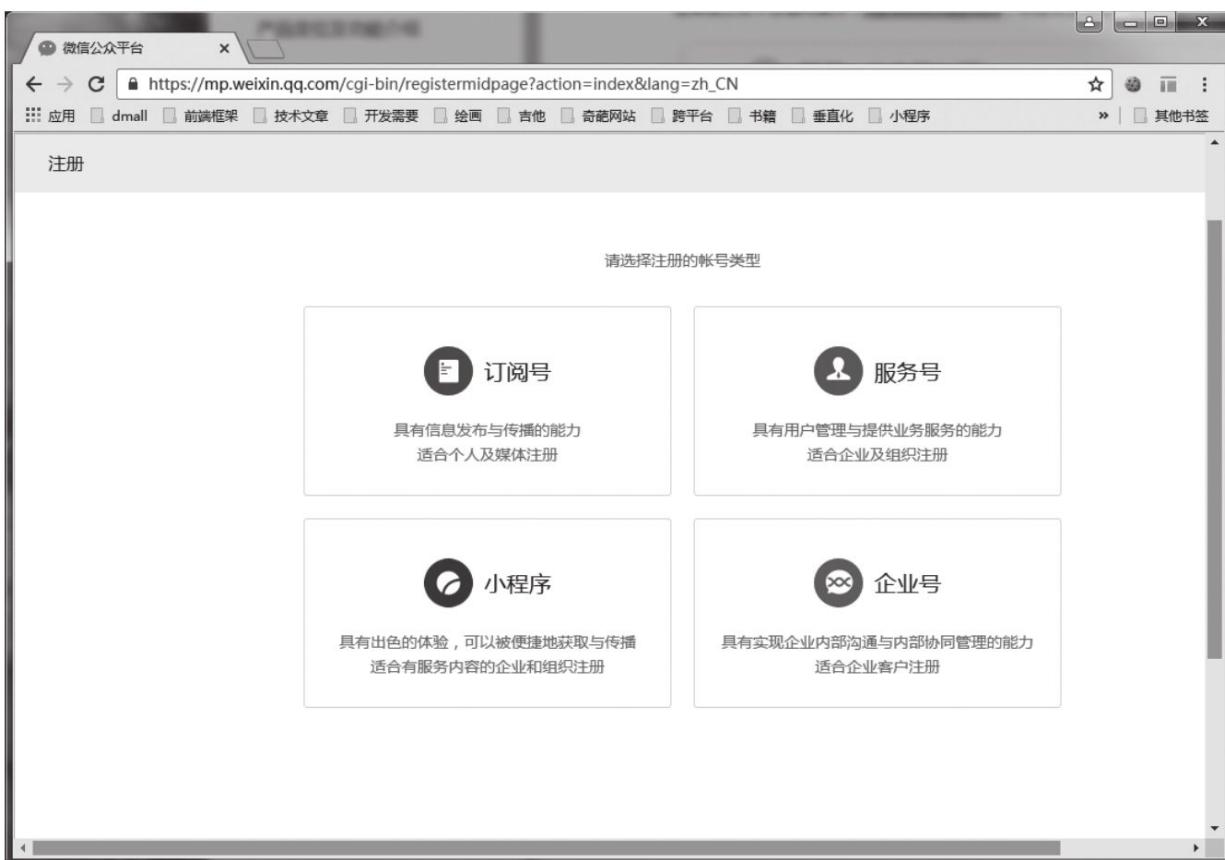


图1-2 选择帐号类型



图1-3 填写邮箱信息

4) 填写个人帐号信息后，会收到一封激活邮件，点击激活链接，进入主体信息页面填写相关内容，即可完成注册流程，主体信息一经提交后便不可修改，如图1-4所示。

微信公众平台

腾讯客服-小程序介绍

https://mp.weixin.qq.com/wxopen/wacontractorpage?action=step3&lang=zh_CN&token=6552265

应用 dmail 前端框架 技术文章 开发需要 绘画 吉他 奇葩网站 跨平台 书籍 垂直化 小程序 其他书签

请确认你的微信公众帐号主体类型属于政府、媒体、企业、其他组织，并请按照对应的类别进行信息登记。
点击查看'微信公众平台信息登记指引'。

主体类型 如何选择主体类型？

企业	政府	媒体	其他组织
----	----	----	------

企业包括：企业、分支机构、个体工商户、企业相关品牌。

主体信息登记

企业类型 ☐ 企业 ☐ 个体工商户

企业名称

需与当地政府颁发的商业许可证书或企业注册证上的企业名称完全一致，信息审核审核成功后，企业名称不可修改。

营业执照注册号

请输入15位营业执照注册号或18位的统一社会信用代码

注册方式 请先填写名称

管理员信息登记

图1-4 完善主体信息

1.2.2 开发环境准备

完成帐户注册后，需要登录微信公共平台官网（mp.weixin.qq.com），根据流程完善小程序信息，如图1-5所示。需要注意的是，目前小程序名称一旦确定后便不能修改。



图1-5 完善小程序信息

开发小程序之前还需要进入“用户身份-开发者”，绑定开发者，如图1-6所示。只有绑定的开发者才能使用开发者工具编写小程序，一个

小程序最多可以绑定20个开发者，未认证的小程序最多可以绑定10个开发者。



图1-6 绑定开发者

添加开发者后，需要进入“设置-开发设置”，获取AppID，如图1-7所示。AppID十分重要，只有填写了AppID的项目才能通过手机微信扫码进行真机测试。



图1-7 获取AppID

最后便可到首页下载开发者工具，如图1-8所示。小程序开发工具主要用于小程序调试、预览，虽然自带了代码编辑功能，但还是建议使用自己熟悉的编辑器对代码进行编辑（如：sublime、atom、brackets等）。



图1-8 下载开发者工具

1.3 第一个小程序

完成开发准备后我们便可以开始编写小程序，微信小程序的开发十分简单，大家可以快速上手。下面我们利用官方提供的dmeo让大家对小程序开发有初步认识。

1) 打开微信开发者工具。第一次启动需要扫描二维码登录，如图1-9所示，开发权限配置参照上一小节。



图1-9 登录微信开发者工具

2) 登录后选择“添加项目”。

3) 在填写项目信息之前，先创建一个空目录作为项目资源目录，这里我们以E: \weixin\demo为例。

4) 填写项目信息。如果没有AppID可以选择“无AppID”；填写项目名称，项目名称在微信开发者工具中是唯一的；项目目录选择刚才创建的空目录，这里一定要保证刚才创建的目录为空目录，这样下面

会出现“在当前目录中创建quick start项目”选项，勾选这个选项，如图1-10所示，然后点击“添加项目”按钮。



< 返回

添加项目

AppID 无 AppID 部分功能受限

返回填写小程序AppID

项目名称 demo

项目目录 E:\weixin\demo 选择

☒ 在当前目录中创建 quick start 项目

取消 添加项目

图1-10 填写项目信息

创建好后的界面如图1-11所示。



图1-11 第一个小程序

这样我们便成功创建了第一个小程序，这个demo是官方提供的示例，第一个页面展示了当前登录的用户信息，点击头像会跳转到一个记录当前小程序启动时间的日志页面。为了让大家进一步体验小程序开发，我们利用Sublime或其他编辑工具打开E: \weixin\demo文件夹，

修改index.wxml，将“{{motto}}”替换为“我的第一个小程序”，
index.wxml修改后代码如下：

```
<!--index.wxml-->
<view class="container">
  <view bindtap="bindViewTap" class="userinfo">
    <image class="userinfo-avatar" src="{{userInfo.avatarUrl}}" background-size=
"cover"></image>
    <text class="userinfo-nickname">{{userInfo.nickName}}</text>
  </view>
  <view class="usermotto">
    <!-- 修改这句代码 -->
    <text class="user-motto">我的第一个小程序</text>
  </view>
</view>
```

修改后点击“重启”，如图1-12所示。

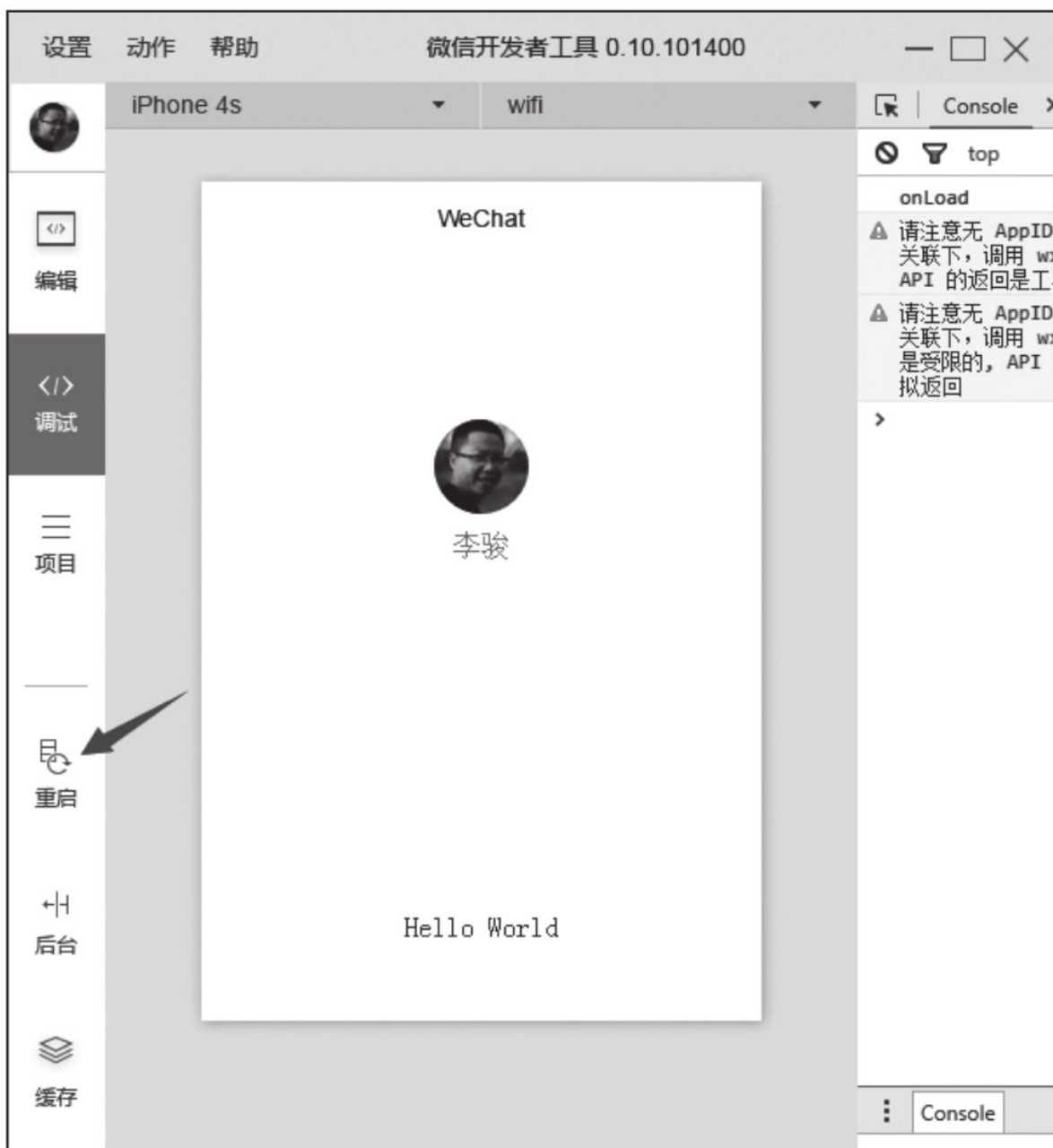


图1-12 重启项目

重启后，登录界面提示语由原来的“Hello World”变成了“我的第一个小程序”，如图1-13所示。



图1-13 我的第一个小程序

填写了AppID的项目可以选择“项目→预览”（如图1-14所示），扫描二维码在微信中体验项目。



图1-14 预览小程序

至此我们简单体验了一个小程序的创建过程，但对于一个喜欢“刨根问底”的学习者来说，这个案例远远不够，可能会有很多问题，例如：

- 小程序启动入口在哪里？

- index.wxml、index.wxss、index.js等文件是否可以重新命名？它们之间的关系是什么？目录结构有怎样的规范？

- WXML、WXSS文件是什么？怎么感觉很像HTML和CSS？

- 小程序开发中有哪些限制？

大家可以先按自己理解尝试修改项目中相关的文件，看看能产生什么效果，把过程中遇到的问题记录下来，带着问题阅读后面的章节。

1.4 小结

本章简单介绍了小程序开发流程，让大家对小程序有体验式理解。小程序的学习过程并不难，掌握框架、组件、**API**后可快速上手，技术细节问题我们将在后续章节中进行详细讲解。本章仅介绍了主要的流程，接入过程中碰到的细节问题可参考官方资料：

<http://mp.weixin.qq.com/debug/wxadoc/introduction>。

第2章 小程序开发核心

上一章讲解了小程序创建流程，本章主要为大家讲解小程序框架及核心内容。小程序框架可让开发者在微信中用尽可能简单、高效的方式开发出具有原生App体验的服务，这套框架控制着小程序完整的生命周期，负责页面的加载、渲染、销毁等工作，它是小程序的核心，学习小程序前，我们一定要对这套框架有深入的了解。本章主要对小程序目录结构、文件类型进行详细分析，重点介绍小程序视图层WXML、MXSS，逻辑层JS，这些是小程序开发的核心内容。本章个别小节内容比较深，学习过程中不必过于深究，能对框架有个整体认识即可。

2.1 简介

小程序框架将整个系统划分为视图层和逻辑层，视图层是由框架设计的标签语言WXML（WeiXin Markup Language）和用于描述WXML组件样式的WXSS（WeiXin Style Sheets）组成，它们的关系就像HTML和CSS的关系。WXML和WXSS在渲染时会被框架解析为不同端的本地渲染文件，这样保证一套代码能在多处运行，并且能最大化地接近原生App。渲染原理和React Native、Weex十分接近，开发过程中我们不必深究WXML的渲染原理，只需要有个大致了解即可。小程序逻辑层是一套运行在本地JavaScript引擎的JavaScript代码，在此基础上框架实现了一套模块化机制，让每个JS文件有独立的作用域和模块化能力，这套模块化机制遵循CommonJS规范，熟悉NodeJs的开发者应该有一定了解。

小程序整体开发流程非常接近前端HTML+CSS+JavaScript的开发模式，与前端开发不同的是，在小程序中没有DOM的概念，在本地的JavaScript引擎中也没有window、document等对象，我们不能想当然地通过操作DOM来操作页面，小程序中的视图层和逻辑层的交互是通过数据绑定和事件响应实现的，这是一种单向绑定的机制。这套机制需要首先将逻辑层和视图层的数据和事件进行绑定，当需要修改页面时，逻辑层只需要调用特定的setData方法修改已绑定的数据，这时框

架会自动触发WXML重新渲染，达到逻辑层对视图层的控制；当框架接收到用户交互操作时，会根据视图层绑定的事件，执行逻辑层中对应的事件函数，达到逻辑层对视图层的响应，视图层与逻辑层的关系如图2-1所示。这套机制是小程序框架的工作原理，在后续内容中我们将反复提及，加深大家对它的理解。

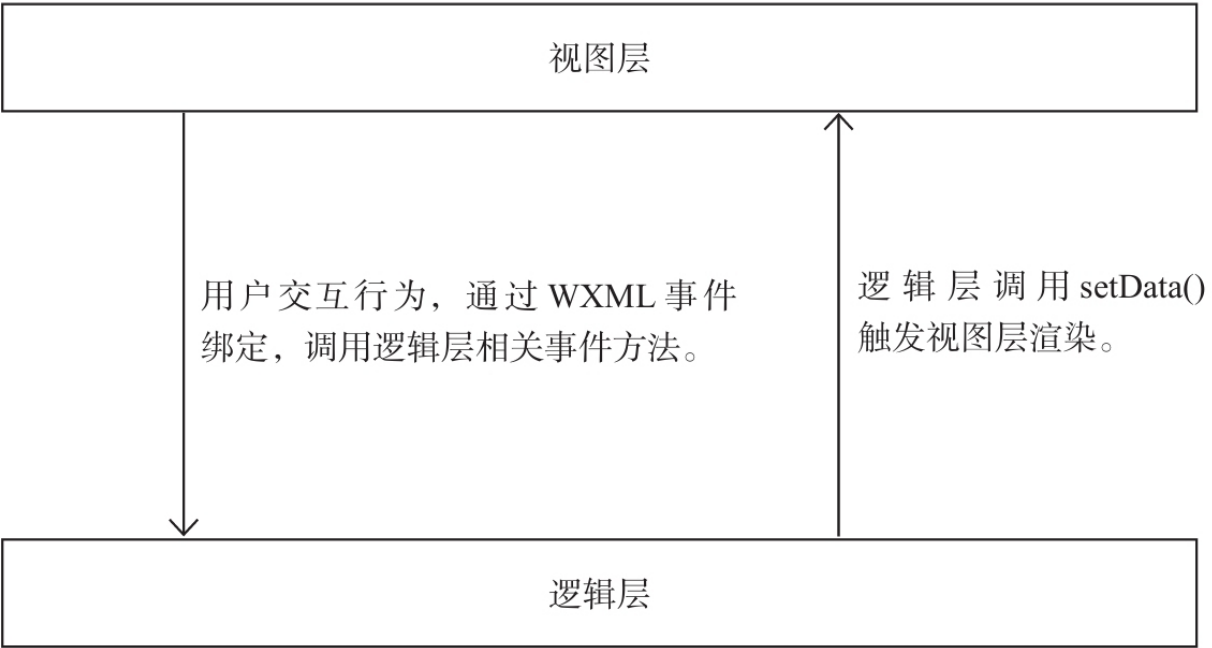


图2-1 视图层与逻辑层关系

2.2 “徒手”创建小程序

为了让开发者更好地理解小程序框架运行机制，接下来将带领大家“徒手”创建一个结构最简单的小程序，这样每个细节都是开发者自己完成的，这对理解小程序框架有很大帮助。步骤如下：

- 1) 创建项目目录，这里以E: \weixin\myproject为例。
- 2) 按图2-2所示的目录结构创建文件：
- 3) 打开app.json，写入以下代码：

```
{
  "pages" : [
    /* 指定默认启动页面地址 */
    "mypages/index/index"
  ]
}
```

- 4) 打开index.wxml，写入以下代码：

```
<view bindtap="countClick">我是index页面,你点击了{{count}}次</view>
```

- 5) 打开index.js文件，写入以下代码：

```
Page( {
  data : {
    count : 0
  },
  countClick : function() {
    this.setData( {
      count : this.data.count + 1
    })
  }
})
```

```
    } );  
  }  
};
```

就这么几步，一个最简单的小程序便搭建好了，项目中仅包含一个index页面，这个目录结构是最简单、最基础的小程序结构。接下来我们将它导入开发者工具中看看运行效果。

6) 打开微信开发者工具，填写AppId和项目名称，点击“选择”按钮添加项目，项目目录选择刚才创建的目录E: \weixin\myproject，点击“添加项目”完成添加，如图2-3所示：

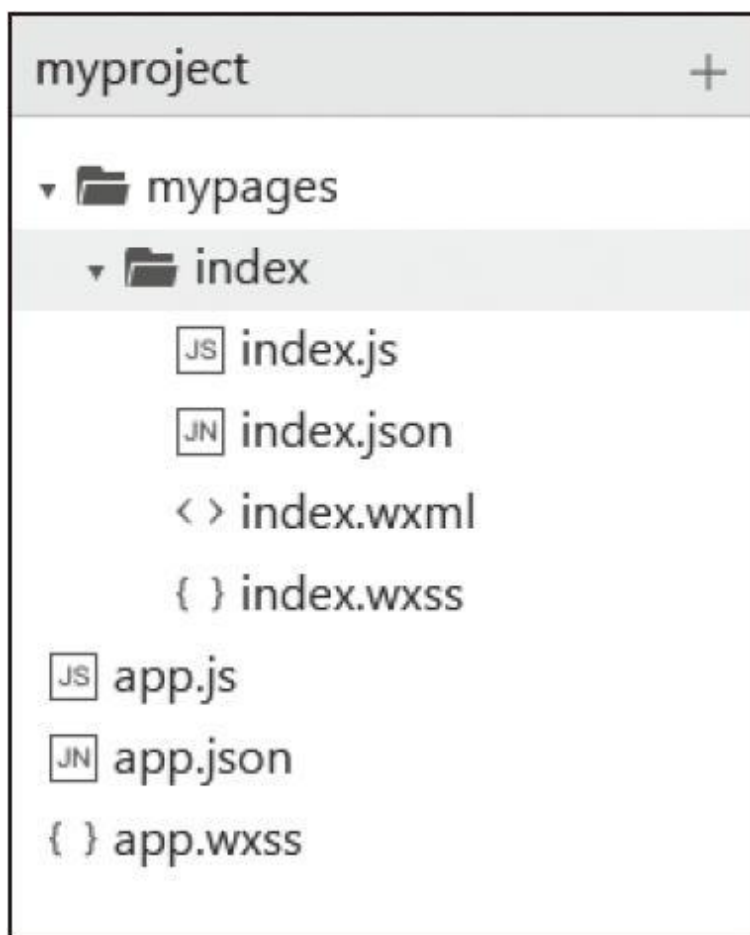


图2-2 目录结构

< 返回

添加项目

AppID 无 AppID 部分功能受限

返回填写小程序AppID

项目名称 myproject

项目目录 E:\weixin\myproject 选择

取消 添加项目

图2-3 项目配置界面

7) 导入项目后我们便能看到运行界面，当我们点击文字时，点击次数也会随之增加（如图2-4所示）。

这就是最简单的小程序，所有复杂的项目都是围绕这个结构进行拓展的。当运行这个项目时，框架首先会解析配置文件app.json，通过pages设置找到默认首页页面mypages/index/index（pages第一个路径默

认为首页），然后加载mypage/index目录中index.wxml、index.wxss、index.js、index.json这4个文件进行页面渲染。

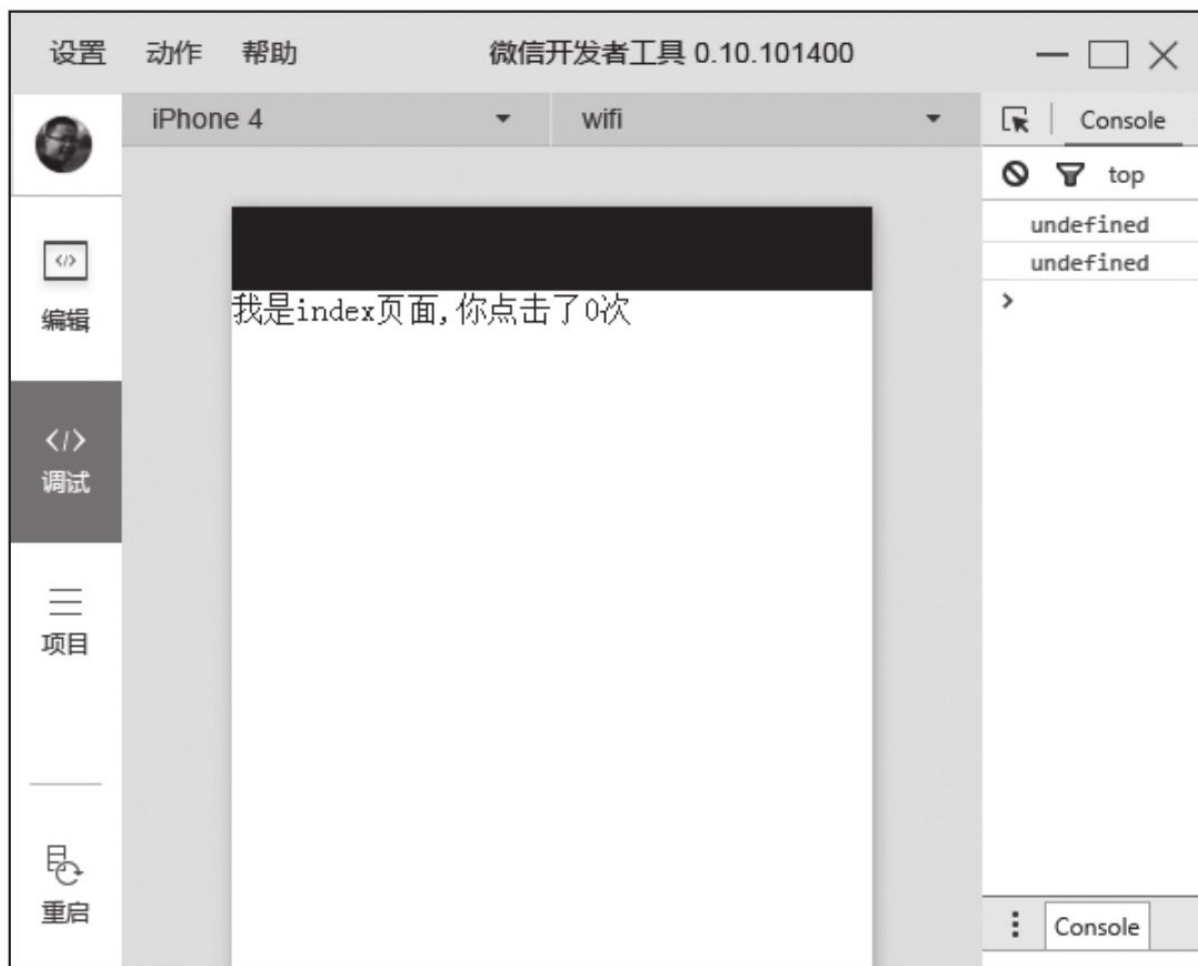


图2-4 myproject运行界面

在index.wxml文件中，我们简单使用了<view/>组件，页面渲染时，框架将逻辑层中data的count属性与视图层的count进行了绑定，所以一打开页面会显示点击次数为0。当点击<view/>时，会触发tap事件，这时视图层根据<view/>组件bindtap属性值，将绑定的countClick事件发送给逻辑层，逻辑层根据方法名找到对应的事件处理函数

`countClick`并执行。`countClick`函数中，我们调用了`setData`方法修改`count`值，并触发视图层渲染，所以页面中的数字随着点击次数增加，这种视图层和逻辑层之间相互通信的机制便是小程序的数据绑定和事件响应系统。

在一个完整的小程序中，文件主要分为框架程序主体文件和页面文件两大类：

- 框架程序主体文件是系统级别文件，一个项目只有一份，分别是`app.json`、`app.js`和`app.wxss`，它们分别控制小程序整体配置、逻辑和整体样式，小程序启动时只会执行一次。这3个文件必须放在项目根目录，且文件名必须是`app`，其中`app.json`和`app.js`是必须的。

- 一个小程序有一个或多个页面，一个页面由`.wxml`、`.wxss`、`.js`和`json`四个文件组成，它们分别控制页面的结构、样式、逻辑和配置，其中`.wxml`文件和`.js`文件是必须的，按照框架规定，同一个页面的这4个文件必须具有相同的路径和文件名，所以在这个项目中我们将它们放置在`mypages/index`路径下且文件名统一为`index`，其中`index`目录名可以和页面文件名不一致，为了便于管理我们尽量将页面目录名和页面文件名保持一致。

现在我们对小程序框架有个大致的认识，下面，将分别为大家讲解框架主体文件和框架页面的特性及使用方法。

2.3 框架主体文件

框架主体文件由`app.json`、`app.js`、`app.wxss`构成，这3个文件必须放置在项目根目录，一个小程序只有一份，它们负责小程序整体的配置：

- `app.json`：小程序公共设置，配置小程序全局设置。

- `app.js`：小程序逻辑文件，主要用于注册小程序全局实例，编译时会和其他页面逻辑文件打包成一份JavaScript文件。

- `app.wxss`：小程序公共样式表，对所有页面的布局文件都有效。

`app.json`和`app.js`是必须存在的，`app.wxss`不是必须创建的，可以根据项目情况进行创建。接下来我们逐个分析每个文件。

2.3.1 配置文件（app.json）

app.json是小程序配置文件，编写时要严格遵循json的格式规范。
app.json在程序加载时加载，负责对小程序的全局配置，其配置项有：

- pages: 设置页面路径，必填项。
- window: 设置默认页面的窗口表现。
- tabBar: 设置tab的表现。
- networkTimeout: 设置网络超时时间。
- debug: 设置是否开启debug模式，默认关闭。

app.json文件内容整体结构如下：

```
{  
  // 页面路径设置  
  "pages" : [],  
  // 默认页面的窗口设置  
  "window" : {},  
  // 底部tab设置  
  "tabBar" : {},  
  // 设置网络请求API的超时时间  
  "networkTimeout" : {},  
  // 是否为debug模式  
  "debug" : false  
}
```

1.pages配置

pages负责注册小程序页面，必须填写，**value**值为一个包含页面路径的数组，用来指定小程序由哪些页面构成，每一项由页面“路径+文件名”组成，如下所示：

```
{
  "pages" : [
    "mypages/index/index",
    "mypages/page2/myfilename",
    "otherpages/index/index"
  ]
}
```

pages数组中页面路径不需要填写文件后缀名，渲染页面时框架会自动寻找路径.json，.js，.wxml，.wxss四个文件进行整合，如上文配置中“mypages/index/index”路径，页面加载时框架会自动匹配寻找“mypages/index/index.json”、“mypages/index/index.js”、“mypages/index/index.wxml”、“mypages/index/index.wxss”这四个文件，路径中的文件名可以和目录名不一致，但在项目过程中，为了便于管理，建议文件名和目录名保持一致。**pages**配置数组第一项代表小程序的初始页面。小程序中增加、删除页面，都需要对**pages**进行修改，并且重启项目。

2.window配置

window负责设置小程序状态栏、导航条、标题、窗口背景色等系统级样式。属性有：

- navigationBarBackgroundColor**: 导航栏背景颜色，值为HexColor（十六进制颜色值），如：“#ff83fa”，默认值为#000000。

·navigationBarTextStyle: 导航栏标题颜色，仅支持black/white，默认值为white。

·navigationBarTitleText: 导航栏标题文字内容。

·backgroundColor: 窗口背景色，值为HexColor（十六进制颜色值），如：“#ff83fa”，默认值为#ffffff。

·backgroundTextStyle: 下拉背景字体、Loading图的样式，仅支持dark/light。

·enablePullDownRefresh: 是否开启下拉刷新，默认为false，开启后，当用户下拉时会触发页面onPullDownRefresh事件。

配置项目如图2-5所示。

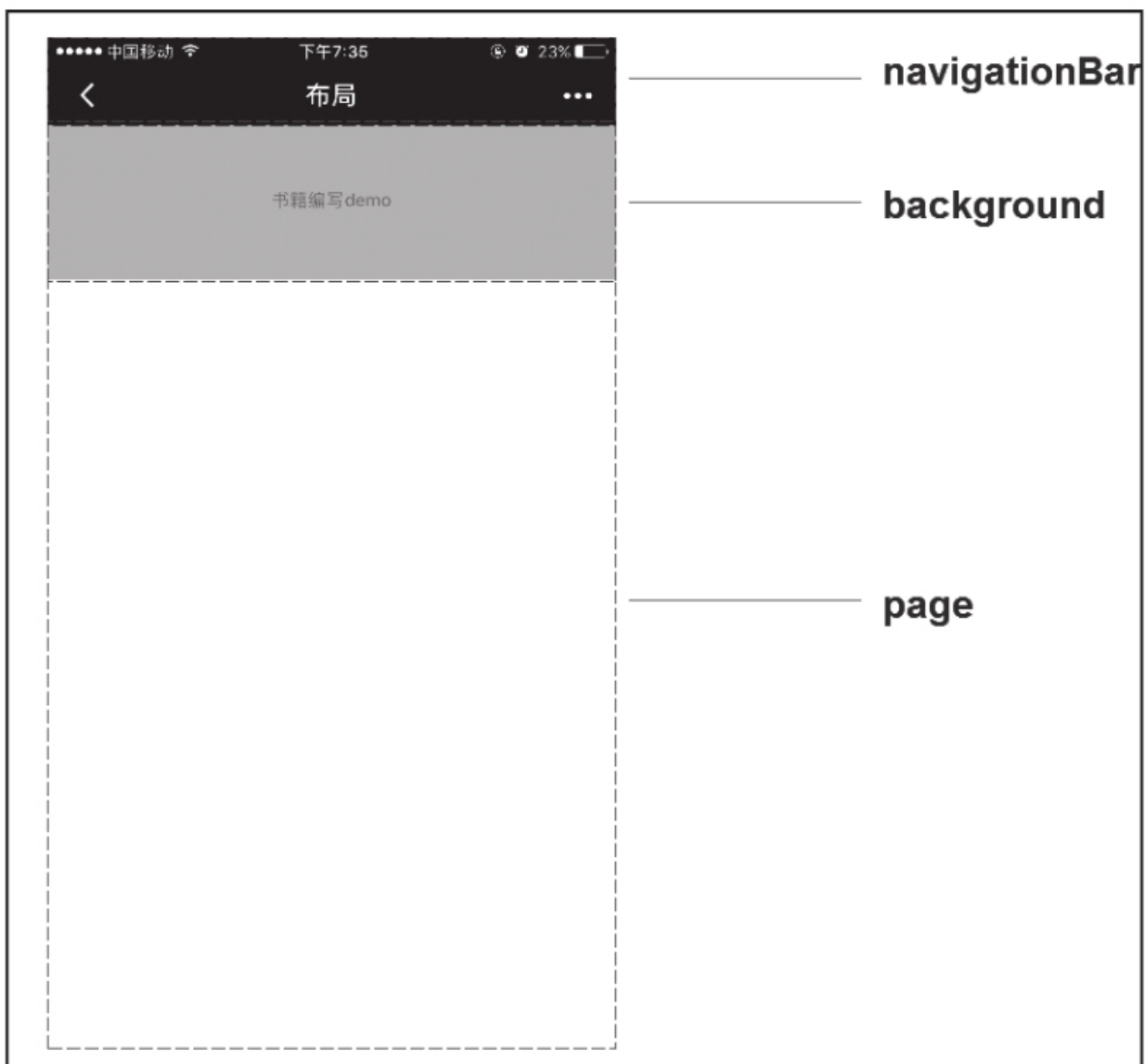


图2-5 界面配置区域

3.tabBar配置

当程序顶部或底部需要菜单栏时，我们可以通过配置tabBar快速实现，tabBar是个非必填项目。可配置属性如下：

·color: tab上的文字默认颜色，值为HexColor（十六进制颜色值），必填项。

·selectedColor: tab上的文字选中时的颜色，值为HexColor（十六进制颜色值），必填项。

·backgroundColor: tab的背景色，值为HexColor（十六进制颜色值），必填项。

·borderStyle: tabbar上边框的颜色，仅支持black/white，默认值为black。

·list: tab的列表，必填项，其值为一个数组，最少2个、最多5个tab，数组中每一项是一个对象，代表一个tab的相关配置，每项的相关配置如下：

·pagePath: 页面路径，必须在pages中先定义，必填项。

·text: tab上按钮的文字，必填项。

·iconPath: tab上icon图片的相对路径，icon大小限制为40kb，必填项。

·selectedIconPath: 选中时图片的相对路径，icon大小限制为40kb，必填项。

·position: tab在顶部或底部显示，可选值为bottom、top，默认值为bottom。

示例代码见代码清单2-1。

代码清单2-1 app.json

```
{
  "pages" : [
    "mypages/index/index",
    "mypages/list/list"
  ],
  "tabBar" : {
    "color" : "#000000",
    "selectedColor" : "#ff7f50",
    "backgroundColor" : "#ffffff",
    "borderStyle" : "black",
    "list" : [
      {
        "iconPath" : "images/home.png",
        "selectedIconPath" : "images/home-selected.png",
        "pagePath" : "mypages/index/index",
        "text" : "首页"
      },
      {
        "iconPath" : "images/search.png",
        "selectedIconPath" : "images/search-selected.png",
        "pagePath" : "mypages/list/list",
        "text" : "搜索"
      },
      {
        "iconPath" : "images/list.png",
        "selectedIconPath" : "images/list-selected.png",
        "pagePath" : "mypages/list/list",
        "text" : "列表"
      }
    ]
  },
  "borderStyle" : "bottom"
}
```

配置后页面效果如图2-6所示。



图2-6 tabBar配置示例

4.networkTimeout配置

小程序中各种网络请求API的超时时间只能通过networkTimeout统一设置，不能在API中单独设置，具体的网络请求API可参考后面章

节，networkTimeout支持的属性有：

- request: 设置wx.request的超时时间，单位毫秒。
- connectSocket: 设置wx.connectSocket的超时时间，单位毫秒。
- uploadFile: 设置wx.uploadFile的超时时间，单位毫秒。
- downloadFile: 设置wx.downloadFile的超时时间，单位毫秒。

示例代码如下：

```
{
  "pages" : [
    "mypages/index/index"
  ],
  "networkTimeout" : {
    "request" : 60000,
    "connectSocket" : 60000
  }
}
```

5.debug配置

此配置项控制是否开启debug模式，默认是关闭的。开启debug模式后，在开发者工具的控制面板，调试信息以info的形式输出，如图2-7所示。其中信息有Page的注册、页面路由、数据更新、事件触发，可以帮助开发者快速定位一些常见问题。

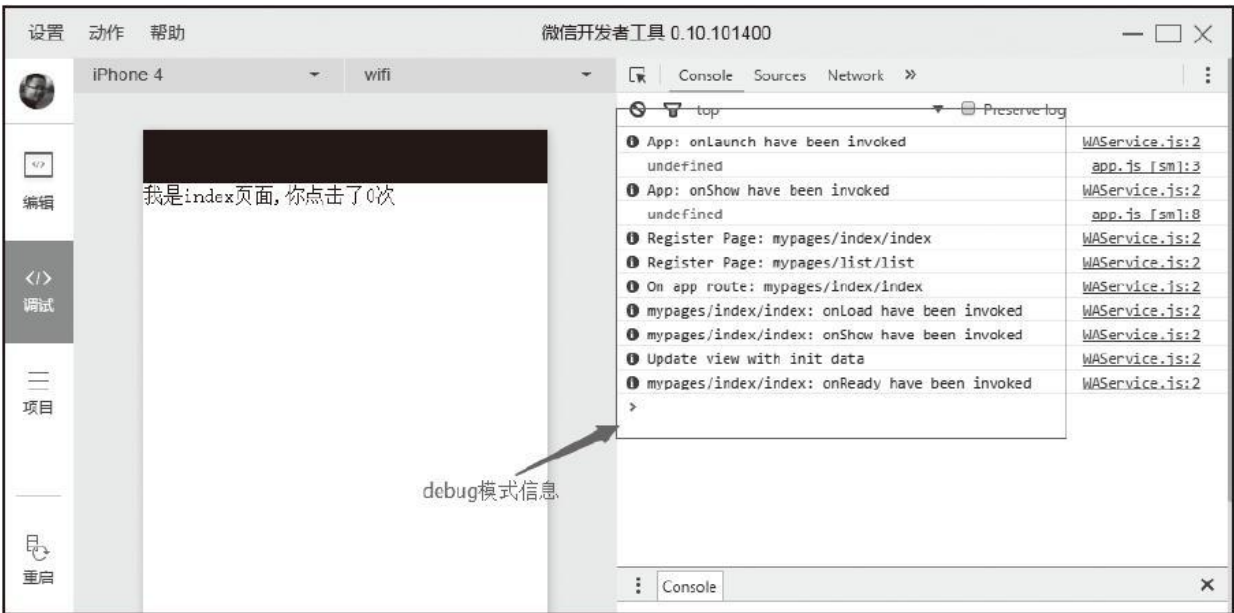


图2-7 开启debug模式

示例代码如下：

```
{
  "pages" : [
    "mypages/index/index",
    "mypages/list/list"
  ],
  "debug" : true
}
```

以上便是app.json的5类配置项，这些配置项都是全局的，小程序中除了app.json这种全局配置文件还有页面配置文件，当路由到对应页面时，页面配置文件的配置项将会覆盖全局配置，页面配置文件将在后续内容中进行详细介绍。

2.3.2 小程序逻辑（app.js）

小程序中逻辑文件分为页面逻辑文件和小程序逻辑文件，`app.js`便是小程序逻辑文件，在这个文件中，我们可以通过**App（）**函数注册小程序生命周期函数、全局方法和全局属性，已注册的小程序实例可以在其他逻辑层代码中通过**getApp（）**获取。

1.注册小程序

App（）函数用于注册一个小程序，参数为一个**Object**对象，在这个参数对象中我们可以注册自定义方法和属性供全局使用，就像在**quick start**项目中，我们利用**App（）**注册了用户登录信息。**App（）**函数必须在**app.js**中注册，且不能注册多个，其参数属性如下：

·**onLaunch**：生命周期函数，监听小程序初始化。当小程序初始化完成时，就会触发**onLaunch**，**onLoaunch**事件全局只会触发一次。

·**onShow**：生命周期函数，监听小程序显示。当小程序启动，或者从后台进入前台显示时都会触发**onShow**。

·**onHide**：生命周期函数，监听小程序隐藏。当小程序从前台进入后台会触发。

·其他：开发者可以添加任意的函数或数据到Object参数中，这些属性会被注册到小程序对象中，其他逻辑文件可以通过getApp () 函数获取已注册的小程序实例。

关于小程序生命周期函数的执行时机我们要特别讲解一下：当启动一个小程序时，首先会先依次触发onLaunch和onShow方法，然后通过app.json的pages属性注册相应的页面，最后根据默认路径加载首页；当用户点击左上角关闭，或者按了设备Home按钮离开微信时，小程序并没有直接销毁，而是进入了后台，这两种情况都会触发onHide方法；当再次唤醒微信（针对点击Home按钮离开微信）或再次从微信中打开小程序时，又会从后台进入前台，这时会触发onShow方法。只有当小程序进入后台一定时间，或者系统资源占用过高，才会被真正销毁。

注册小程序示例代码如下：

```
App( {  
  onLaunch : function() {  
    // 小程序初始化完成时执行  
  },  
  onShow : function() {  
    // 显示小程序时执行  
  },  
  onHide : function() {  
    // 隐藏小程序时执行  
  },  
  globalFunction : '我是全局函数',  
  globalData : '我是全局属性'  
} );
```

2.获取小程序实例

注册小程序后，在其他逻辑文件中，可以通过全局函数`getApp`
() 获取小程序实例，例如：

```
var app = getApp();  
console.log( app.globalData );
```

在`App` () 注册的函数中，我们可以使用`this`直接获取`App`实例，而不用`getApp` () 方法。通过`getApp` () 获取实例后，可以获取注册的属性、调用注册的方法，但不要私自调用生命周期函数

(`onLaunch`、`onShow`、`onHide`)，这样会打乱项目逻辑，除非你已经对它们很熟悉。

2.3.3 全局样式 (app.wxss)

`app.wxss`是全局样式表，对项目中每个页面都有效，可将一些系统级别的统一样式风格写入这个文件，页面渲染时，框架页中的`.wxss`文件样式会覆盖`app.wxss`中相同的选择器样式。WXSS是小程序基于CSS拓展的一套样式语言，它实现了CSS大部分规则，其具体介绍请参考下一节。

小程序框架主体相关的文件`app.json`、`app.js`、`app.wxss`我们已经全部介绍完成，下面为大家介绍框架页面相关的文件。

2.4 框架页面文件

小程序中一个框架页面包含4个文件，同一框架页面的这4个文件必须具有相同的路径与文件名，进入小程序时或页面跳转时，小程序会根据app.json配置的路径找到对应的资源进行渲染。

·.js文件：页面逻辑文件，必要项。

·.wxml文件：页面结构文件，必要项。

·.wxss文件：页面样式文件。

·.json文件：页面配置文件。

与框架主体文件相比框架页面文件多了一种页面结构文件，其余3个文件和框架主体文件的功能类同，下面我们一一讲解每个文件作用。

2.4.1 页面配置文件

我们首先讲解页面配置文件，页面配置文件和框架配置文件一样，是一个json文件，与框架配置文件不同的是，页面配置文件是非必要存在的，同时页面配置文件的配置项只有window，控制当前页面的窗口表现，window的属性和app.json一致。渲染页面时，页面中的window配置项会覆盖app.json中的相同配置项。

由于页面的.json只能配置window相关属性，编写时只需直接写出属性，不用写window这个键，如下所示：

```
{
  "navigationBarBackgroundColor": "#000000",
  "navigationBarTextStyle": "black",
  "navigationBarTitleText": "我的页面",
  "backgroundColor": "#efefef",
  "backgroundTextStyle": "light"
}
```

2.4.2 页面逻辑文件（JavaScript）

页面逻辑文件主要功能有：设置初始化数据，注册当前页面生命周期函数，注册事件处理函数等。小程序的逻辑层文件是JavaScript文件，所有的逻辑文件，包括app.js，最终将会打包成一个js文件，在小程序启动时运行，直到小程序销毁，类似于ServiceWorker，所以逻辑层也称为App Service。

在小程序中，每个逻辑文件有独立的作用域，并具备模块化能力。由于JavaScript逻辑文件是运行在纯JavaScript引擎中而并非运行在浏览器中，一些浏览器提供的特有对象，如document、window等，在小程序中都无法使用，同理，一些基于document、window的框架如：jQuery、Zepto都不能在小程序中使用，同时我们不能通过操作DOM改变页面，这时需要我们将面向DOM操作的编程思路转化为数据绑定和事件响应。

1.注册页面

在页面逻辑文件中需要通过Page（）函数注册页面，指定页面的初始数据、生命周期函数、事件处理函数等，参数为一个Object对象，其属性如下：

·**data**: 页面的初始数据，数据格式必须是可转成JSON格式的对象类型。当页面第一次渲染时，**data**会以JSON的形式由逻辑层传至渲染层，渲染层可以通过WXML对数据进行绑定。

·**onLoad**: 生命周期函数，页面加载时触发。一个页面只会调用一次，接受页面参数，可以获取wx.navigateTo、wx.redirectTo以及中的query参数。

·**onShow**: 生命周期函数，页面显示时触发。每次打开页面都会调用一次。

·**onReady**: 生命周期函数，页面初次渲染完成时触发。一个页面生命周期中只会调用一次，代表当前页面已经准备妥当，可以和视图层进行交互。一些对界面的设置，操作需要在页面准备妥当后调用，如wx.setNavigationBarTitle需要在onReady之后设置，详细可以参考后面的“页面生命周期”小节。

·**onHide**: 生命周期函数，页面隐藏时触发。

·**onUnload**: 生命周期函数，页面卸载时触发。

·**onPullDownRefresh**: 页面相关时间处理函数，用户下拉时触发。使用时需要将app.json配置中window的enablePullDownRefresh属性设置

为true。当处理完数据刷新后，可以调用wx.stopPullDownRefresh方法停止当前页面的下拉刷新。

- onReachBottom: 页面上拉触底事件的处理函数。

- 其他: 开发者可以添加任意的函数或数据到Object参数中，可以用this访问这些函数和数据。

示例代码如代码清单2-2所示:

代码清单2-2 页面逻辑文件

```
// 获取app实例
var app = getApp();
Page( {
  data : {      // 页面初始化数据
    count : 0
  },
  onLoad : function() {
    // 页面加载时执行
  },
  onShow : function() {
    // 页面打开时执行
    console.log( app.globalData );
  },
  onReady : function() {
    // 页面初次渲染完成执行，一个页面只会调用一次
  },
  onHide : function() {
    // 页面隐藏时执行
  },
  onUnload : function() {
    // 页面卸载时执行
  },
  onPullDownRefresh : function() {
    // 下拉刷新时执行
  },
  onReachBottom : function() {
    // 下拉触底时执行
  },
  // 自定义函数，可与渲染层中的组件进行事件绑定
  countClick : function() {
    // 触发视图层重新渲染
    this.setData( {
      count : this.data.count + 1
    } );
  },
  // 自定义数据
```

```
customData : {  
  name : '微信'  
}  
} );
```

页面的生命周期函数比小程序的生命周期函数略微复杂一点，弄懂其执行顺序能避免在不恰当的生命周期函数中调用还未创建的对象或方法，小程序框架以栈的形式维护了当前的所有页面，当发生路由切换时，页面栈和生命周期函数的关系如下：

- 小程序初始化：默认页面入栈，依次触发默认页面onLoad、onShow、onReady方法。

- 打开新页面：新页面入栈，依次触发新页面onLoad、onShow、onReady方法。

- 页面重定向：当前页面出栈并卸载，触发当前页面onUnload方法，新页面入栈，触发新页面onLoad、onShow、onReady方法。

- 页面返回：页面不断出栈并卸载，触发当前弹出页面onUnload方法，直到返回目标页面，新页面入栈，触发新页面onShow方法。

- Tab切换：当前页面出栈但不卸载，仅触发onHide方法，新页面入栈，如果当前页面是新加载的，触发onLoad、onShow、onReady方法，如果当前页面已加载过，仅触发onShow方法。

·程序从前台到后台：触发当前页面onHide方法，触发App onHide方法。

·程序从后台到前台：触发小程序onShow方法，触发页面onShow方法。

整体来说，如果页面在生命周期中，只会触发onShow和onLoad方法，只有加载和卸载时才会触发onLoad、onReady和onUnload方法，而触发页面卸载只有页面返回和页面重定向两种操作。页面生命周期函数的执行顺序不用死记硬背，开发过程中可以开启app.json中的debug模式，路由切换时，小程序、页面的生命周期函数执行顺序都会在控制台以info形式输出，在后面小节中我们将对页面生命周期作简单介绍。

2.获取当前页面栈

有注册就有获取，getCurrentPages () 函数便是用于获取当前页面栈的实例，页面栈以数组形式按栈顺序给出，第一个元素为首页，最后一个元素为当前页面。不要尝试修改页面栈，这会导致路由以及页面状态错误。

示例代码如下：

```
/* 获取页面栈 */  
var pages = getCurrentPages();
```

```
/* 获取当前页面对象 */  
var currentPage = pages[pages.length - 1];
```

3.事件处理函数

页面对象中注册的函数可以和视图层中的组件进行绑定，当达到触发条件时，就会执行Page中定义的相应事件，这类自定义函数统称为事件处理函数。小程序中组件的事件分为通用事件和特殊事件，事件类型请参考后面2.4.3节中“事件”小节。

示例代码如下：

```
<view bindtap= "myevent ">点击执行逻辑层事件</view>  
  
Page( {  
  myevent : function() {  
    console.log( '点击了view' );  
  }  
} );
```

4.触发视图层渲染

页面首次加载时，框架会结合初始化数据渲染页面，在逻辑层中则需主动调用Page.prototype.setData () 方法，而不能直接修改Page的data值，这样不仅无法触发视图层渲染，还会造成数据不一致。当Page.prototype.setData () 被调用时，会将数据从逻辑层发送到视图层触发视图层重绘，同时会修改Page的data值。setData () 接受一个Object对象参数，方法会自动将this.data中的key对应的值变成Object参数中key对应的值。当Object参数key对应的值和this.data中key对应的值

一致时，将不会触发视图层渲染。在项目中我们一定要保证视图层和逻辑层的数据一致。

Object参数的key值非常灵活，可以按数据路径的形式给出，如array[5].info、obj.key.subkey，并且这样使用时，不需要在this.data中预先定义。

示例代码如下：

```
<view>{{text}}</view>
<button bindtap="changeText">修改普通数据</button>

<view>{{object.subObject.objectText}}</view>
<button bindtap="changeObjectText">修改对象数据</button>

<view>{{array[0].arrayText}}</view>
<button bindtap="changeArrayText">修改数组数据</button>

<view>{{newField.newFieldText}}</view>
<button bindtap="addNewData">添加新字段</button>
Page( {
  data : {
    text : 'normal data',
    object : {
      subObject : {
        objectText : 'object data'
      }
    },
    array : [
      { arrayText : 'array data' }
    ]
  },
  changeText : function() {
    this.setData( {
      /* 普通索引 */
      text : 'new normal data'
    } );
  },
  changeObjectText : function() {
    this.setData( {
      /* 按路径索引 */
      'object.subObject.objectText' : 'new object data'
    } );
  },
  changeArrayText : function() {
    this.setData( {
      /* 按路径索引 */
      'array[0].arrayText' : 'new array data'
    } );
  },
  addNewData : function() {
```

```
    this.setData( {  
      /* 修改一个已绑定，但未在data中定义的数据 */  
      'newField.newFieldText' : 'add new data'  
    } );  
  }  
} );
```

5. 页面生命周期

页面的生命周期整体关系着页面视图层线程和页面逻辑层线程，注册页面时，**Object**参数中很多属性都是生命周期函数，这些函数的调用和页面生命息息相关，程序视图层线程和逻辑层线程关系如图2-8所示。

图2-8 Page实例的生命周期

如图2-8，线程启动后视图层和逻辑层相互监听，当逻辑层线程触发onLoad、onShow方法后会把初始数据data传送给视图层线程，视图层完成第一次渲染后触发逻辑层onReady方法，代表页面已经准备妥当，之后我们便可通过setData方法主动触发视图层渲染。当页面被调往后台时，触发onHide方法，这时逻辑层线程并没有销毁，我们仍然可以通过代码控制视图层渲染，只是可能不会在界面上表现出来。当页面从后台回到前台时，触发onShow方法，最后当页面销毁时，触发onUnload方法。整体来看onLoad、onReady和onUnload方法在生命周期中只会调用一次，生命周期内显示、隐藏页面都是触发onShow和onHide方法，在路由方式中，只有页面重定向和页面返回会结束当前页面生命周期，当进入一个已加载的页面时只会触发onShow方法，不会触发onLoad和onReady方法。

2.4.3 页面结构文件（WXML）

WXML（WeiXin Markup Language）是框架设计的一套标记语言，用于渲染界面，WXML的渲染原理和React Native思路一致，通过一套标记语言，在不同平台被解析为不同端的渲染文件，如图2-9所示。

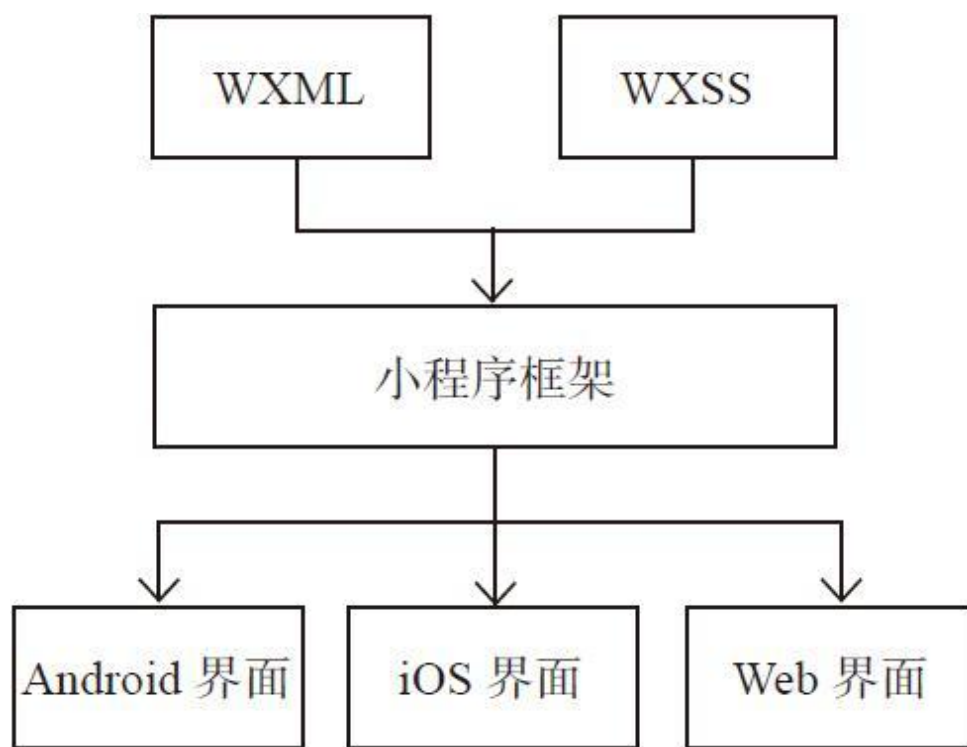


图2-9 界面渲染示意图

从图中我们能看出，WXML语言最终会转译为宿主端对应的语言，所以WXML中所使用的标签一定是小程序定义的标签，不能使用自定义标签，这样才能保证页面能被正确转译。使用微信开发者工具开发时，在WXML中编写一些HTML标签或自定义标签仍然会被正常

解析，这会给开发者造成一种小程序能直接支持HTML标签的误解。这是因为微信开发者工具内核是浏览器内核，同时小程序框架并没对WXML中的标签和WXSS中的内容进行强验证，所以HTML和CSS能直接被解析，但这种不合法的WXML在手机端微信中是不能正常显示的。开发过程中我们一定要拿真机进行测试，保证程序能正常运行。

WXML具有数据绑定、列表渲染、条件渲染、模板、事件等能力。

1.数据绑定

小程序中页面渲染时，框架会将WXML文件同对应Page的data进行绑定，在页面中我们可以直接使用data中的属性。小程序的数据绑定使用Mustache语法（双大括号）将变量或简单的运算规则包起来，主要有以下几种渲染方式。

（1）简单绑定

简单绑定是指我们使用Mustache语法（双大括号）将变量包起来，在模板中直接作为字符串输出使用，可作用于内容、组件属性、控制属性、关键字等输出，其中关键字输出是指将JavaScript中的关键字按其真值输出。

示例代码如下：

```
<!-- 作为内容 -->
<view>{{content}}</view>

<!-- 作为组件属性 -->
<view id="item-{{id}}" style="border:{{border}}">作为属性渲染</view>

<!-- 作为控制属性 -->
<view wx:if="{{showContent}}">作为属性渲染</view>

<!-- 关键字 -->
<view>{{2}}</view>
<checkbox checked="{{false}}"></checkbox>

Page( {
  data : {
    border : 'solid 1px #000',
    id : 1,
    content : '内容',
    showContent : false
  }
} );
```

运行效果如图2-10所示。

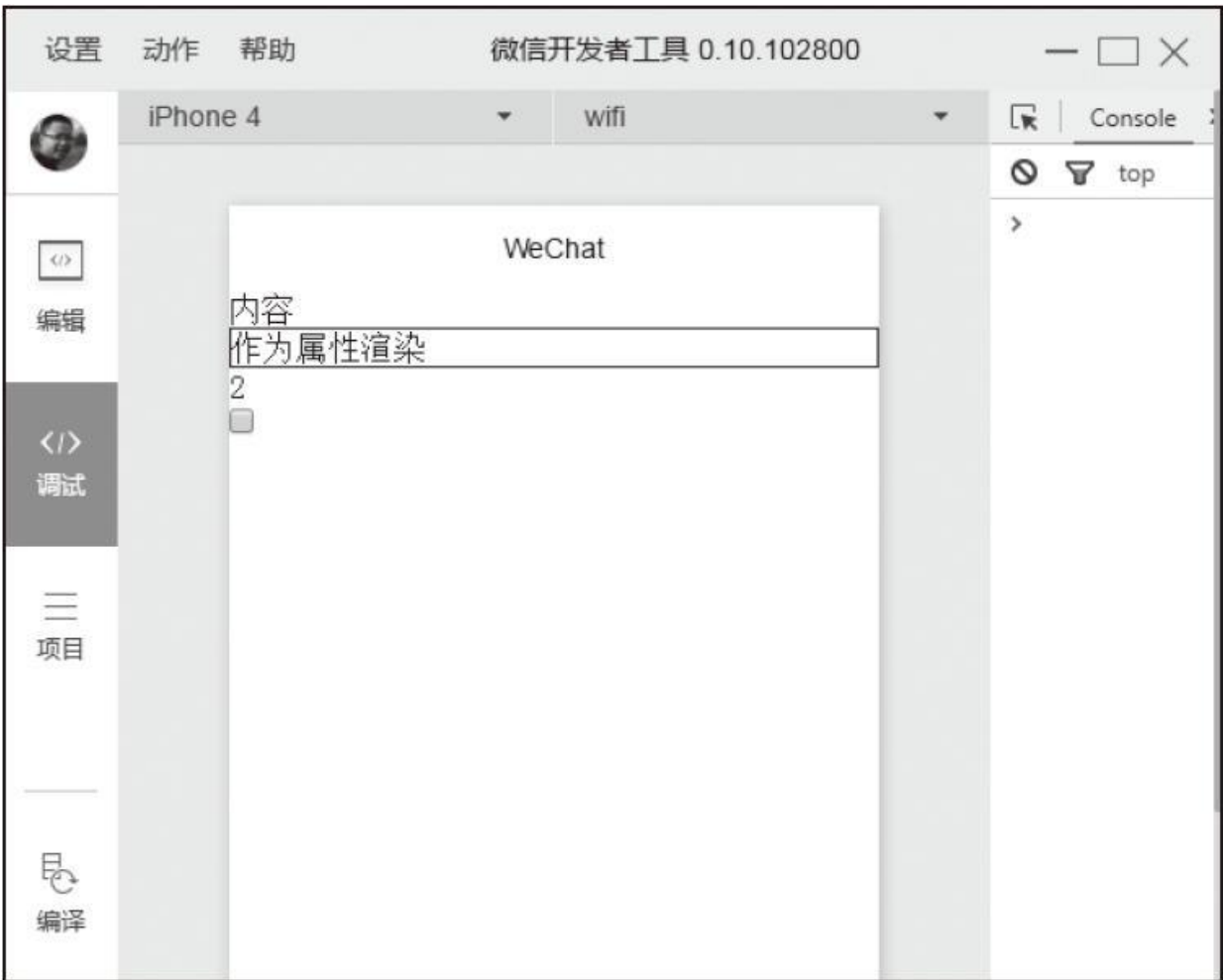


图2-10 简单绑定渲染效果

注意

组件属性为boolean类型时，不要直接写checked="false"，这样checked的值是一个false的字符串，转成boolean类型后代表为true，这种情况一定要使用关键字输出：checked="{{false}}"

(2) 运算

在`{{}}`内可以做一些简单的运算，支持的运算有三元运算、算数运算、逻辑判断、字符串运算，这些运算均符合JavaScript运算规则。我们利用如下示例为大家展示：

```
<!-- 三元表达式 -->
<view>{{ showContent ? '显示文本' : '不显示文本'}}</view>
<!-- 算数运算符 -->
<view>{{ num1 + num2 }} + 1 + {{ num3 }} = ? </view>

<!-- 字符串运算 -->
<view>{{ "name : " + name }}</view>

<!-- 逻辑判断 -->
<view>{{ num3 > 0 }}</view>

<!-- 数据路径运算 -->
<view>{{ myObject.age }} {{myArray[1]}}</view>

Page( {
  data : {
    showContent : false,
    num1 : 1,
    num2 : 2,
    num3 : 3,
    name : 'weixin',
    myObject : {
      age : 12
    },
    myArray : ['arr1','arr2']
  }
} );
```

执行后界面如图2-11所示。



图2-11 运算示例

(3) 组合

data中的数据可以在模板再次组合成新的数据结构，这种组合常常在数组或对象中使用。

数组组合 比较简单，可以直接将值放置到数组某个下标下：

```
<view>{{ [myValue, 2, 3, 'stringtype'] }}</view>
```

```
Page( {  
  data : {
```

```
    myValue : 0
  }
} );
```

最终页面组合成的对象为[0, 2, 3, 'stringtype']。

对象组合 有3种组合方式，这里我们以数据注入模板为例。

第一种，直接将数据作为**value**值进行组合：

```
<template is="testTemp" data="{ { name : myvalue1, age : myvalue2 } }"></template>

Page( {
  data : {
    myValue1 : 'value1',
    myValue2 : 'value2'
  }
} );
```

最终组合出的对象为{name: 'value1', age: 'value2'}。

第二种，通过“...”将一个对象展开，把key-value值拷贝到新的结构中：

```
<template is="testTemp"
  data="{ { ...myObj1, key5 : 5, ...myObj2, key6 : 6 } }">
</template>

Page( {
  data : {
    myObj1 : {
      key1 : 1,
      key2 : 2
    },
    myObj2 : {
      key3 : 3,
      key4 : 4
    }
  }
} );
```

最终组合成的对象为{key1: 1, key2: 2, key5: 5, key3: 3
key4: 4, key6: 6}

第三种，如果对象key和value相同，可以只写key值：

```
<template is="testTemp" data="{{ key1, key2 }}"></template>

Page( {
  data : {
    key1 : 1,
    key2 : 2
  }
} );
```

这种写法最后组合成的对象是{key1: 1, key2: 2}

上述3种方式可以根据项目灵活组合，要注意的是和js中的对象一样，如果一个组合中有相同的属性名时，后面的属性将会覆盖前面的属性，如：

```
<template is="testTemp" data="{{...myObj, key1 : 3}}"></template>

Page({
  data : {
    key1 : 1,
    key2 : 2
  }
});
```

示例中key1是重复的属性，那么后面的属性将会覆盖前面的属性，最终组合生成的对象为{key1: 3, key2: 2}。

2.条件渲染

(1) wx: if

除了简单的数据绑定，我们常常会使用逻辑分支，这时候可以使用`wx: if="{{判断条件}}"`来进行条件渲染，当条件成立时渲染该代码块：

```
<view wx:if="{{showContent}}">内容</view>

Page( {
  data : {
    showContent : false
  }
} );
```

示例中`view`代码块将不会渲染，只有当`showContent`的值为`true`时才渲染。

和普通的编程语言一样，WXML也支持`wx: elif`和`wx: else`，如：

```
<view wx:if="{{false}}">1</view>
<view wx:elif="{{false}}">2</view>
<view wx:else>3</view>
```

示例中页面只渲染最后一个`<view/>`。`wx: elif`和`wx: else`必须和`wx: if`配合使用，否则会导致页面解析出错，在项目中大家一定要注意。

(2) block wx: if

`wx: if`是一个控制属性，可以添置在任何组件标签上，但如果我们需要包装多个组件，又不想影响布局，这时就需要使用`<block/>`标签将需要包装的组件放置在里面，通过`wx: if`作判断。`<block/>`不是一个组

件，仅仅是一个包装元素，页面渲染过程中不做任何渲染，由属性控制，如下所示：

```
<block wx:if="{{true}}">
  <view>view组件</view>
  <image/>
</block>
```

(3) wx: if与hidden

除了wx: if组件，也可以通过hidden属性控制组件是否显示，开发者难免有疑问，这两种方式该怎样取舍，这里我们整理了两种方式的区别：

- wx: if控制是否渲染条件块内的模板，当其条件值切换时，会触发局部渲染以确保条件块在切换时销毁或重新渲染。wx: if是惰性的，如果在初始渲染条件为false时，框架将什么也不做，在条件第一次为真时才局部渲染。

- hidden控制组件是否显示，组件始终会被渲染，只是简单控制显示与隐藏，并不会触发重新渲染和销毁。

综合两个渲染流程可以看出，由于wx: if会触发框架局部渲染过程，在频繁切换状态的场景中，会产生更大的消耗，这时尽量使用hidden；在运行时条件变动不大的场景中我们使用wx: if，这样能保证页面有更高效率的渲染，而不用把所有组件都渲染出来。

3.列表渲染

(1) wx: for

组件的wx: for控制属性用于遍历数组，重复渲染该组件，遍历过程中当前项的下标变量名默认为index，数组当前项变量名默认为item，如：

```
<view wx:for="{{myArray}}">
  {{index}}:{{item}}
</view>

Page( {
  data : {
    myArray : [ 'value1', 'value2' ]
  }
} );
```

通过遍历myArray，页面渲染了两个<view/>，结果如图2-12所示。

(2) wx: for-index和wx: for-item

index、item变量名可以通过wx: for-index、wx: for-item属性修改，如：

```
<view wx:for="{{myArray}}" wx:for-index="myIndex" wx:for-item="myItem">
  {{myIndex}}:{{myItem.name}}
</view>
Page( {
  data : {
    myArray : [
      { name : 'value1' },
      { name : 'value2' }
    ]
  }
} );
```

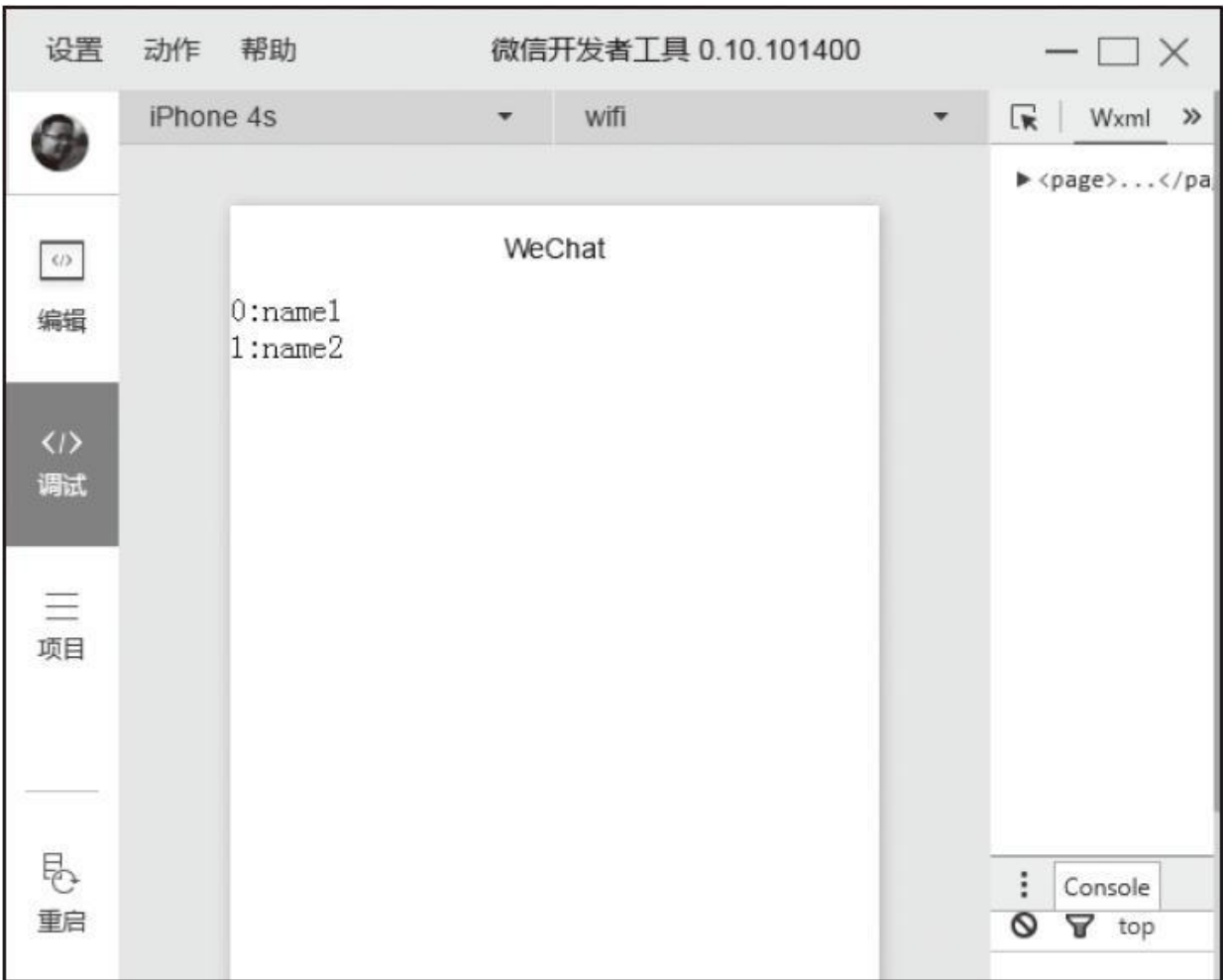


图2-12 列表渲染示例1

渲染结果如图2-13所示。

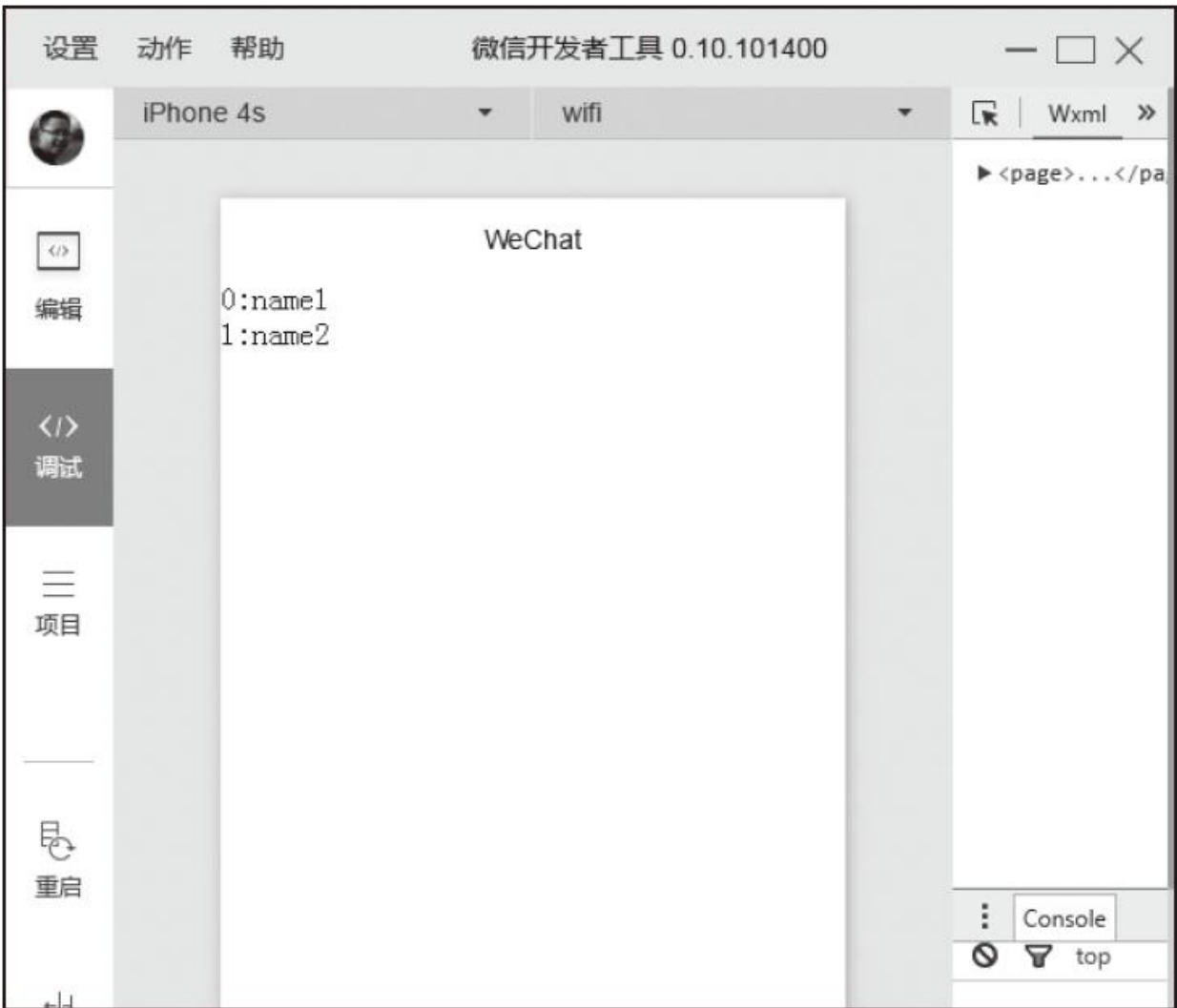


图2-13 列表渲染示例2

普通遍历中我们没必要修改`index`、`item`变量名，当`wx: for`嵌套使用时，就有必要设置变量名，避免变量名冲突，下面我们遍历一个二维数组：

```
<view wx:for="{{myArray}}" wx:for-index="myIndex" wx:for-item="myItem">
  <block wx:for="{{myItem}}" wx:for-index="subIndex" wx:for-item="subItem" >
    {{subItem}}
  </block>
</view>

Page( {
  data : {
```

```
myArray : [  
  [1, 2, 3],  
  [4, 5, 6],  
  [7, 8, 9]  
]  
}  
} );
```

渲染效果如图2-14所示。

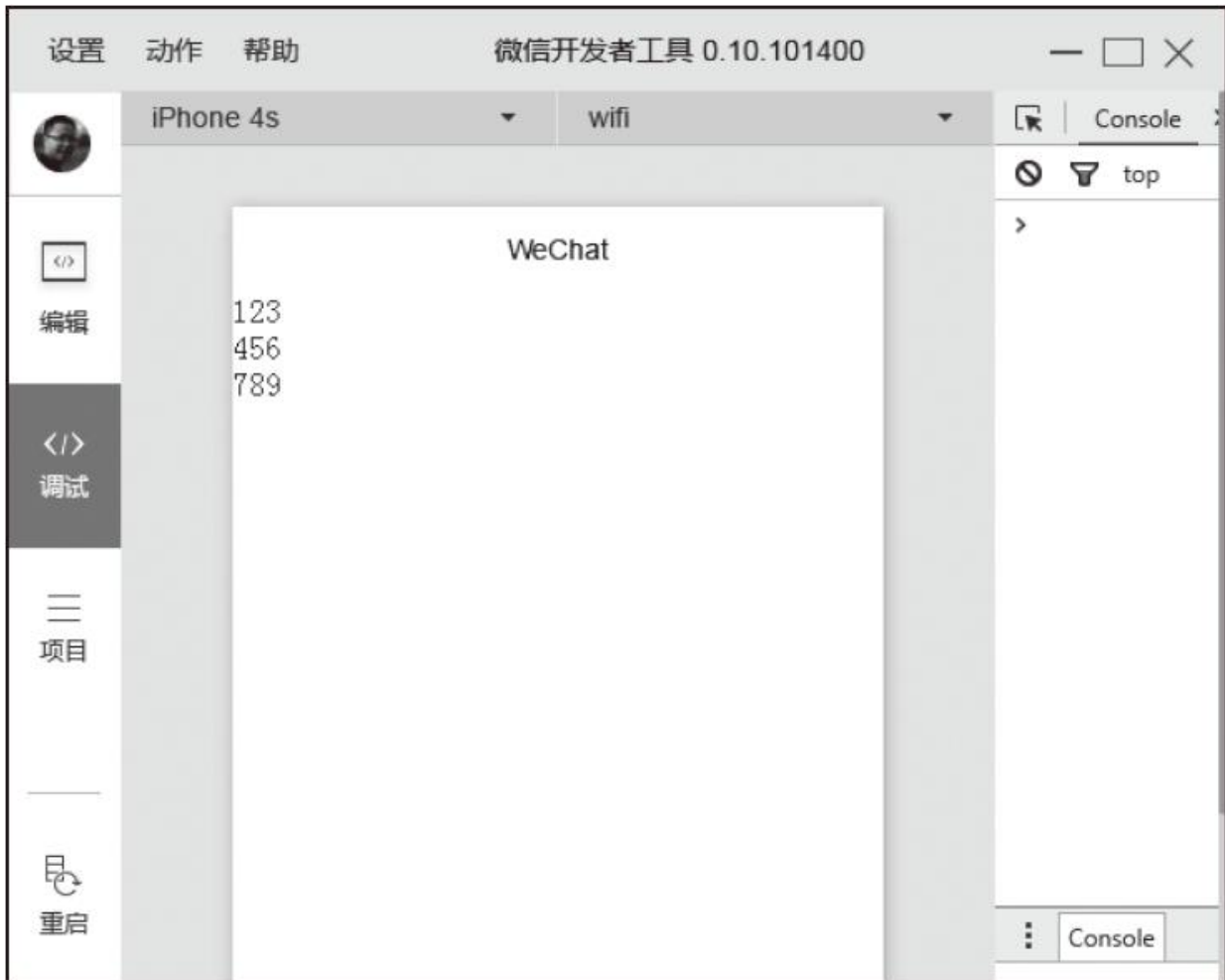


图2-14 列表渲染示例3

在本示例中，我们使用了<block/>标签，和block wx: if一样，wx: for可以直接在<block/>标签上使用，以渲染一个包含多个节点的

结构块。

4.模板

在项目过程中，常常会遇到某些相同的结构在不同的地方反复出现，这时可以将相同的布局代码片段放置到一个模板中，在不同的地方传入对应的数据进行渲染，这样能避免重复开发，提升开发效率。

(1) 定义模板

定义模板非常简单，在<template/>内定义代码片段，设置<template/>的name属性，指定模板名称即可。如：

```
<template name="myTemplate">
  <view>内容</view>
  <view>{{content}}</view>
</template>
```

(2) 使用模板

使用模板时，设置is属性指向需要使用的模板，设置data属性，将模板所需的变量传入。模板拥有自己的作用域，只能使用data属性传入的数据，而不是直接使用Page中的data数据，渲染时，<template/>标签将被模板中的代码块完全替换。

示例代码如下：

```
<template name="myTemplate">
  <view>内容</view>
```



```
<view>{{content}}</view>
<view>{{name}}</view>
<view>{{myObj.key1}}</view>
<view>{{key2}}</view>
</template>
<template is="myTemplate" data="{{content : '内容', name, myObj, ...myObj2}}"/>

Page( {
  data : {
    name : 'myTemplate',
    myObj : {
      key1 : 'value1'
    },
    myObj2 : {
      key2 : 'value2'
    }
  }
} );
```

执行效果如图2-15所示。

模板可以嵌套使用，如下所示：

```
<template name="bTemplate">
  <view>b tempalte content</view>
</template>
<template name="aTemplate">
  <view>a template content</view>
  <template is="bTemplate"/>
</template>
<template is="aTemplate"/>
```



图2-15 模板示例

渲染结果如图2-16所示。

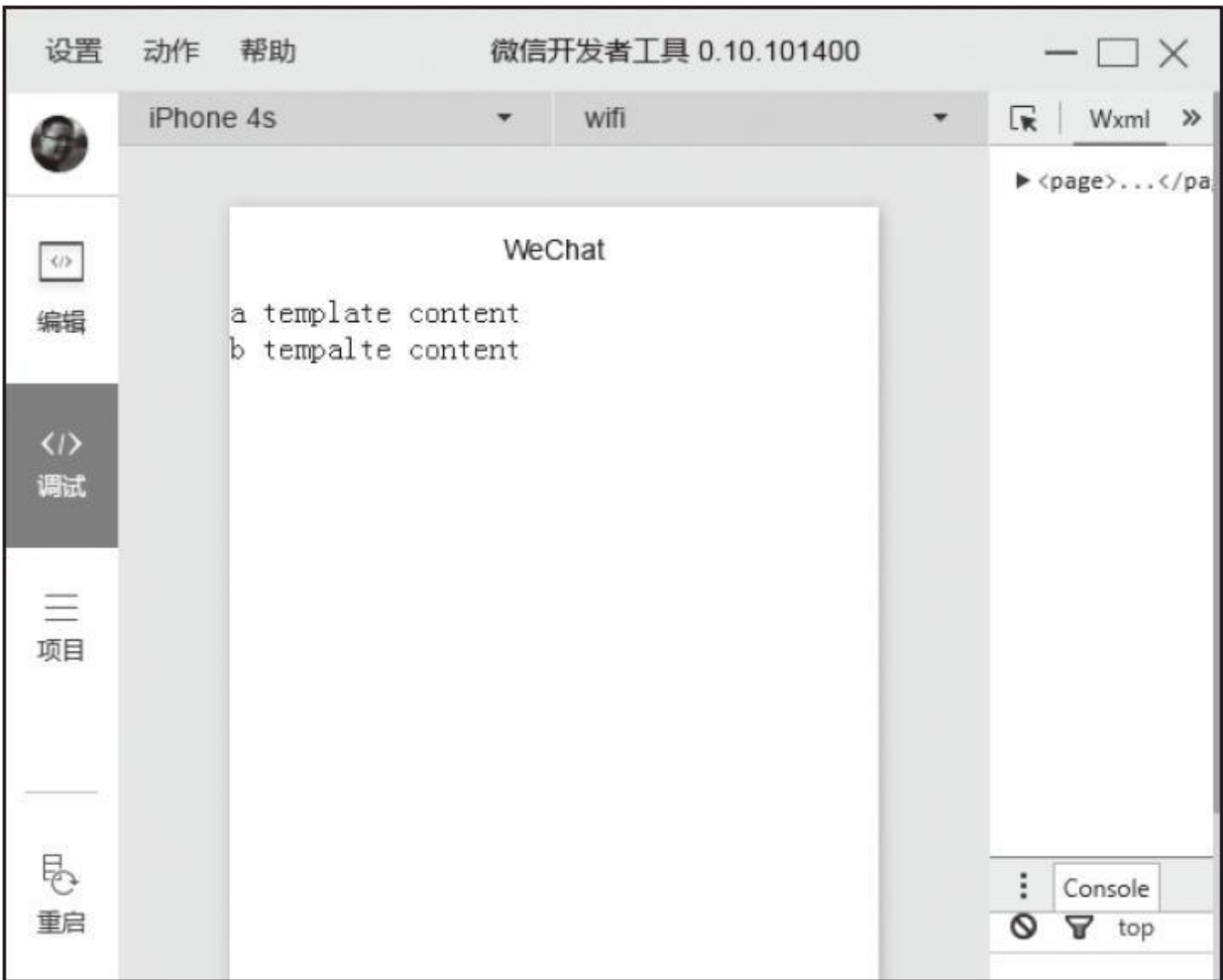


图2-16 嵌套使用模板

注意

模板is属性支持数据绑定，在项目过程中我们可以通过属性绑定动态决定使用哪个模板，如：

```
<template is="{{templateName}}" data="myData"/>
```

5.事件

WXML中的事件系统和HTML中DOM事件系统极其相似，也是通过在组件上设置“bind（或catch）+事件名”属性进行事件绑定，当触发事件时，框架会调用逻辑层中对应的事件处理函数，并将当前状态通过参数传递给事件处理函数，由于小程序中没有DOM节点概念，所以事件只能通过WXML绑定，不能通过逻辑层动态绑定。官方对WXML事件的定义如下：

- 事件是视图层到逻辑层的通讯方式。
- 事件可以将用户的行为反馈到逻辑层进行处理。
- 事件可以绑定在组件上，当触发事件时，就会执行逻辑层中对应的事件处理函数。
- 事件对象可以携带额外信息，如id、dataset、touches。

（1）事件分类

事件分为冒泡事件和非冒泡事件：

- 冒泡事件**： 当一个组件上的事件被触发后，该事件会向父节点传递。
- 非冒泡事件**： 当一个组件上的事件被触发后，该事件不会向父节点传递。

有前端开发经验的开发者应该对事件冒泡都有一定了解，当一个事件被触发后，该事件会沿该组件向其父级对象传播，从里到外依次执行，直到节点最顶层，这个是个非常有用的特性，通常用于实现事件代理，具体实现方案将在下文中具体讨论。

WXML冒泡事件如下：

- touchstart: 手指触摸动作开始。
- touchmove: 手指触摸后移动。
- touchcancel: 手指触摸动作被打断，如来电提醒、弹窗。
- touchend: 手指触摸动作结束。
- tap: 手指触摸后马上离开。
- longtap: 手指触摸后，超过350ms再离开。

对于冒泡事件每个组件都是默认支持的，除上述事件之外的其他组件自定义事件如无特殊声明都是非冒泡事件，如：<form/>的submit事件，<scroll-view/>的scroll事件，详细信息请参考各组件文档。

(2) 事件绑定

在之前内容中，已经多次实现事件绑定，大家应该比较熟悉了，事件绑定的写法和组件的属性一样，以key、value形式组织。

·key: 以bind或catch开头，然后跟上事件类型，字母均小写，如：bindtap，catchtouchstart。

·value: 事件函数名，对应Page中定义的同名函数。找不到同名函数会导致报错。

绑定时bind事件绑定不会阻止冒泡事件向上冒泡，catch事件绑定会阻止冒泡事件向上冒泡。

冒泡示例如下：

```
<view bindtap="tap1">
  view1
  <view catchtap="tap2">
    view2
    <view bindtap="tap3">
      view3
    </view>
  </view>
</view>
```

如上述示例中，点击view3时会先后触发tap3和tap2事件，由于view2通过catch阻止了tap事件冒泡，这时tap1将不会执行，点击view2只触发tap2，点击view1只触发tap1。

(3) 事件对象

如果没有特殊说明，当组件触发事件时，逻辑层绑定该事件的事件处理函数会收到一个事件对象，如：

```
<view bindtap="myevent">view</view>

Page( {
  myevent : function( e ) {
    console.log( e );
  }
} );
```

上述代码中，**myevent**参数**e**便是事件对象，这和JavaScript事件绑定特别像。上述代码执行后事件对象输出如下：

```
{
  "type": "tap",
  "timeStamp": 6571,
  "target": {
    "id": "",
    "offsetLeft": 0,
    "offsetTop": 0,
    "dataset": {
    }
  },
  "currentTarget": {
    "id": "",
    "offsetLeft": 0,
    "offsetTop": 0,
    "dataset": {
    }
  },
  "detail": {
    "x": 15,
    "y": 11
  },
  "touches": [
    {
      "identifier": 0,
      "pageX": 15,
      "pageY": 11,
      "clientX": 15,
      "clientY": 11
    }
  ],
  "changedTouches": [
    {
      "identifier": 0,
      "pageX": 15,
      "pageY": 11,
      "clientX": 15,
      "clientY": 11
    }
  ]
}
```

```
}  
  ]  
}
```

事件对象属性基本可分为三类： **BaseEvent** 、 **CustomEvent** 、 **TouchEvent** 。

BaseEvent 为基础事件对象属性，包括：

·**type**: 事件类型。

·**timeStamp**: 事件生成时的时间戳，页面打开到触发所经过的毫秒数。

·**target**: 触发事件源组件（即冒泡开始的组件）的相关属性集合，属性如下：

·**id**: 事件源组件的id。

·**tagName**: 事件源组件的类型。

·**dataset**: 事件源组件上由data-开头的自定义属性组成的集合。

·**currentTarget**: 事件绑定的当前组件的相关属性集合，属性如下：

·**id**: 当前组件的id。

·**tagName**: 当前组件的类型。

·dataset: 当前组件上由data-开头的自定义属性组成的集合。

<canvas/>中的触摸事件不可冒泡，所以没有currentTarget。

dataset是组件的自定义数据，通过这种方式可以将组件的自定义属性传递给逻辑层。书写方式为：以data-开头，多个单词由连字符“-”连接，属性名不能有大写（大写最终会被转为小写），最终在dataset中将连字符转成驼峰形式，如：

```
<view bindtap="myevent" data-my-name="weixin" data-myAge="12">
  dataset 示例
</view>

Page( {
  myevent : function( e ) {
    console.log( e.currentTarget.dataset );
  }
} );
```

最后dataset打印出来为：

```
{
  "myName" : "weixin", // 连字符被转成驼峰
  "myage" : "12" // 所有大写字符都被转为小写
}
```

CustomEvent 为自定义事件对象（继承BaseEvent），只有一个属性：

·detail: 额外信息，通常传递组件特殊信息。

detail没有统一的格式，在<form/>的submit方法中它是{"value": {}, "formId": ""}，在<swiper/>的change事件中它是{"current":

current}，具体内容参考组件相关文档。

TouchEvent 为触摸事件对象（继承**BaseEvent**）属性如下所示：

- touches**: 触摸事件，当前停留在屏幕中的触摸点信息的数组。

- changedTouches**: 触摸事件，当前变化的触摸点信息的数组，如从有变无（**touchstart**）、位置变化（**touchmove**）、从有变无（**touchend**、**touchcancel**）。

由于支持多点触摸，所以**touches**和**changedTouches**都是数组格式，每个元素为一个**Touch**对象（**canvas**触摸事件中为**CanvasTouch**对象）。

Touch对象相关属性如下：

- identifier**: 触摸点的标识符。

- pageX**, **pageY**: 距离文档左上角的距离，文档的左上角为原点，横向为**X**轴，纵向为**Y**轴。

- clientX**, **clientY**: 距离页面可显示区域（屏幕除去导航条）左上角的距离，横向为**X**轴，纵向为**Y**轴。

CanvasTouch对象相关属性如下：

- identifier**: 触摸点的标识符。

·x, y: 距离Canvas左上角的距离, Canvas的左上角为原点, 横向为X轴, 纵向为Y轴。

6. 引用

一个WXML可以通过import或include引入其他WXML文件, 两种方式都能引入WXML文件, 区别在于import引入WXML文件后只接受模板的定义, 忽略模板定义之外的所有内容, 而且使用过程中有作用域的概念。与import相反, include则是引入文件中除<template/>以外的代码直接拷贝到<include/>位置, 整体来说import是引入模板定义, include是引入组件。

(1) import

<import/>的src属性是需要被引入文件的相对地址, <import/>引入会忽略引入文件中<template/>定义以外的内容, 如下例中, 在a.wxml引入b.wxml, b.wxml中<view/>和<template is="bTemplate"/>都被忽略, 仅引入了模板的定义, 在a.wxml中能使用b.wxml中定义的模板:

```
<import src="b.wxml"/>
<template is="bTemplate" data=""/> <!-- 使用b.wxml中定义的模板 -->

<view>内容</view> <!-- import引用时会被忽略 -->
<template name="bTemplate">
  <view>b template content</view>
</template>
<template is="bTemplate"/> <!-- import引用时会被忽略 -->
```

上述代码中, a.wxml中的<view/>并没有被渲染, 如图2-17所示。

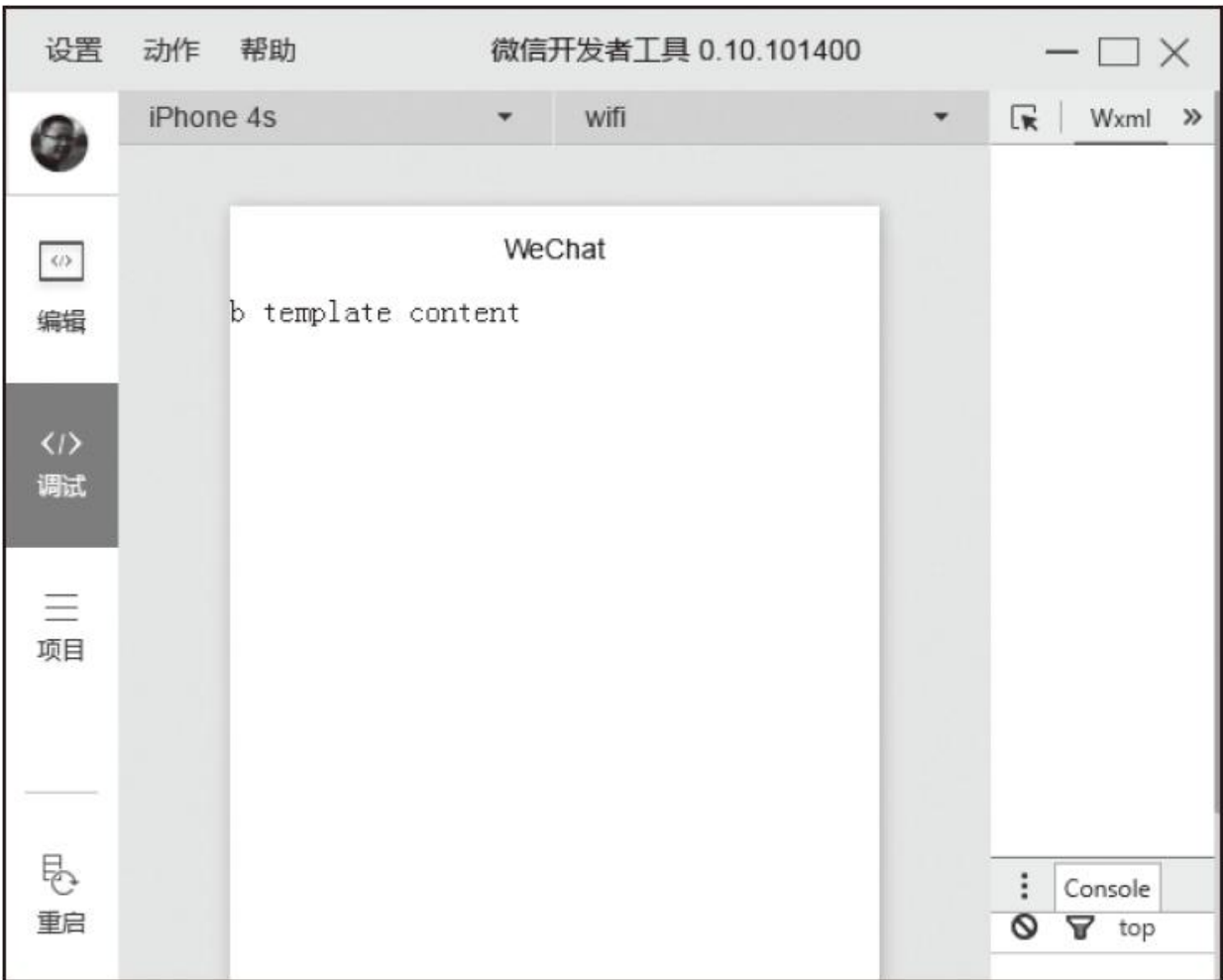


图2-17 import示例1

import引用有作用域概念，只能直接使用引入的定义模板，而不能使用间接引入的定义模板，如下例，在a.wxml中引入b.wxml，b.wxml再引入c.wxml，这样a能直接使用b中定义的模板，b能使用c中定义的模板，但a不能使用c中的模板：

```
<import src="b.wxml"/>
<template is="bTemplate"/>
<template is="cTemplate"/> <!-- 不能直接调用c.wxml中的模板 -->

<import src="c.wxml"/>
<view>b content</view> <!-- import时被忽略 -->
<template name="bTemplate">
```

```
<template is="cTemplate"/>
<view>b tempalte content</view>
</template>
<template is="cTemplate"/> <!-- import时被忽略 -->

<template name="cTemplate">
  <view>c template content</view>
</template>
```

渲染效果如图2-18所示。

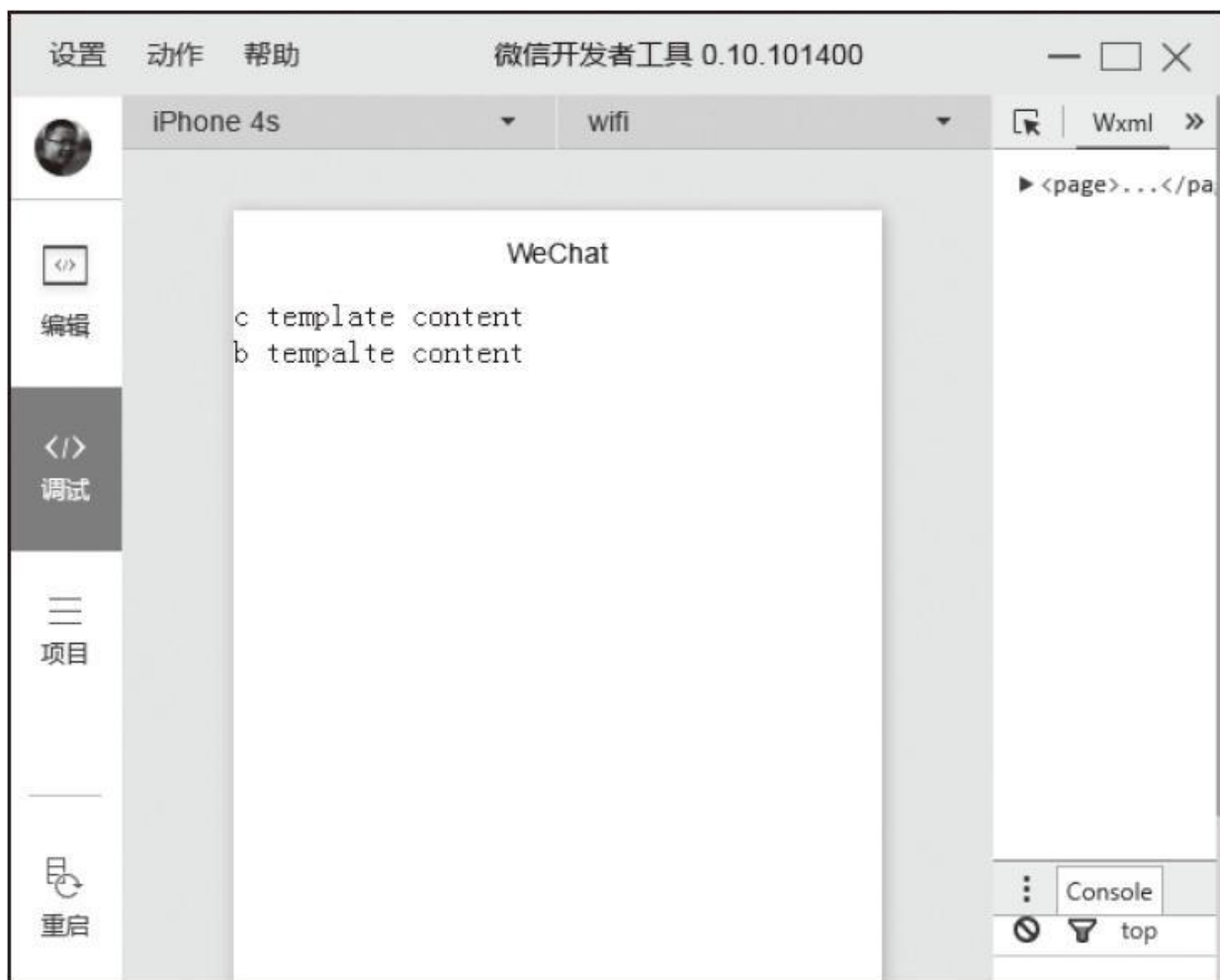


图2-18 import作用域示例

(2) include

include引入会将模板定义标签外的内容（含模板使用标签）直接赋值替换<include/>，我们基于上个案例进行修改，大家对比一下：

```
<include src="b.wxml"/>

<template is="bTemplate"/> <!-- 不能调用b.wxml中的模板 -->
<template is="cTemplate"/> <!-- 不能调用c.wxml中的模板 -->

<include src="c.wxml"/>

<view>b content</view> <!-- 不会被忽略 -->

<template name="bTemplate">
  <template is="cTemplate"/> <!-- 不会调用c.wxml中的模板，引用时已被忽略 -->
  <view>b tempalte content{{name}}</view>
</template>

<template is="bTemplate" data="{{name}}"/> <!-- 没有被忽略，能正常调用自己文件中的模板 -
->

<template name="cTemplate">
  <view>c template content</view>
</template>

Page( {
  data : {
    name : '2' /*能将数据注入到b.wxml中*/
  }
} );
```

运行效果如图2-19所示。

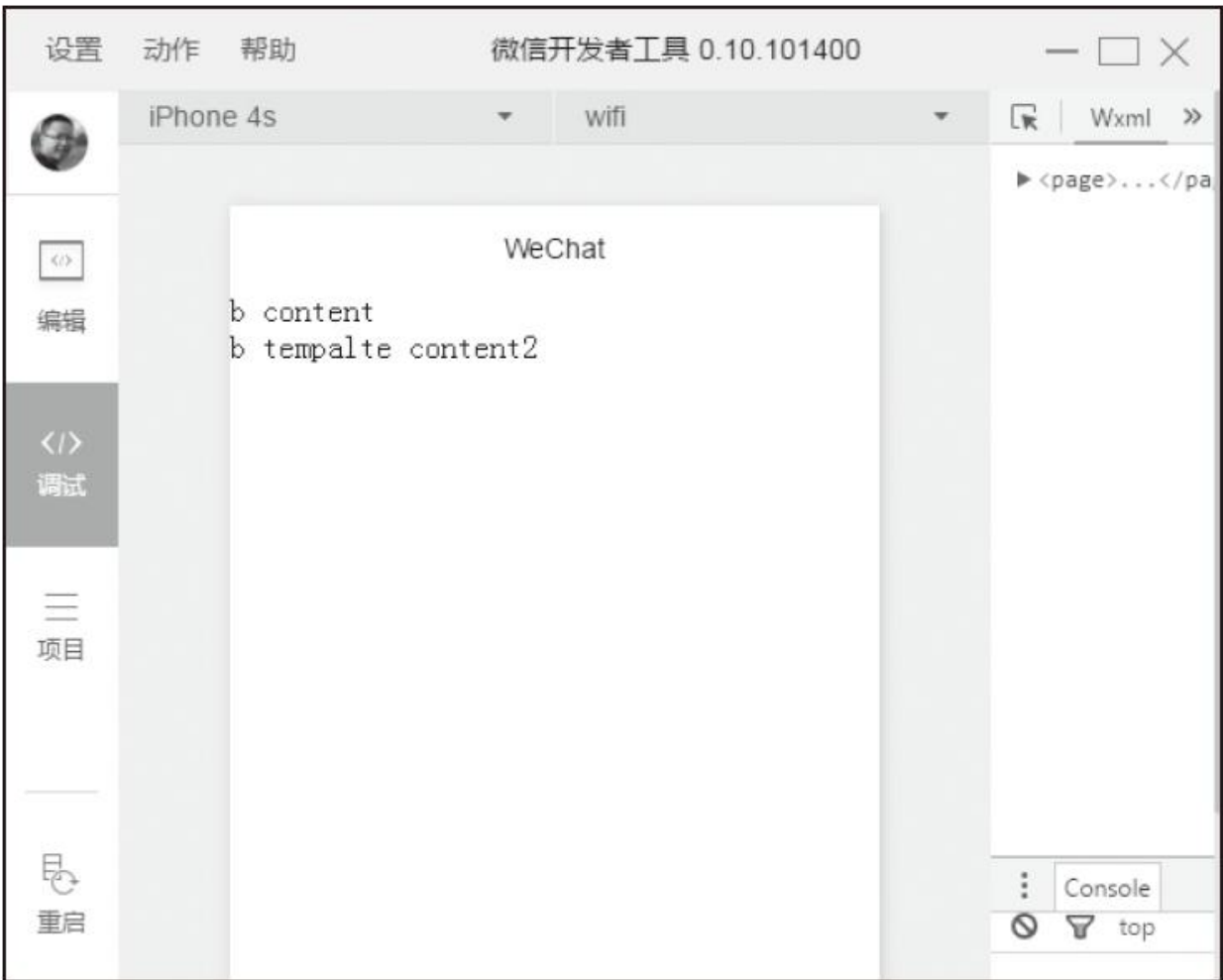


图2-19 include引入示例

通过对比发现，**import**更适合引用模板定义文件，**include**更适合引入组件文件，在项目中大家可以根据特性灵活使用。

WXML虽然是一门新标签语言，但大部分规则和其他前端模板语言大同小异，本节**WXML**规则整体可分为数据绑定，事件机制，模板语法（条件渲染、列表渲染），页面引用（引用规则、模板），大家可以对比其他模板语言学习。

2.4.4 页面样式文件（WXSS）

WXSS（WeiXin Style Sheets）是基于CSS拓展的样式语言，用于描述WXML的组件样式，决定WXML的组件该怎么显示，它具有CSS的大部分特性，在CSS基础上WXSS拓展了尺寸单位、样式导入特性，对CSS选择器属性上做了部分兼容，目前官方文档没有给出WXSS具体具备CSS哪些特性，在开发过程中不能想当然地使用CSS属性，一定要使用iOS和Android真机进行调试，本小节主要讲述WXSS和CSS的不同点，后续布局章节会讲解CSS盒子模型布局相关属性，其余CSS属性大家可以参考W3C规范，在WXSS中我们甚至能使用一些兼容性写法，不过在开发过程中我们一定要开启开发者工具中“开启上传代码时样式文件自动补全”功能，这样小程序会自动补全其余一些样式的兼容性写法，保证在不同终端达到统一视觉效果。

1. 尺寸单位

CSS中原有尺寸单位在不同尺寸屏幕中不能完美实现元素按比例缩放，WXSS在此基础上拓展了两种尺寸单位：rpx（responsive pixel）和rem（root em），这两种尺寸单位都是相对单位，最终会被换算成px，使用rpx和rem布局页面能让页面界面在不同尺寸屏幕中按比例缩放。

（1）rpx

在渲染过程中rpx会按比例转化为px，WXSS规定屏幕宽度为750rpx，如在iPhone6中，屏幕宽度为375px，即750rpx=375px，那么在iPhone6中1rpx=0.5px。

(2) rem

同rpx一样，WXSS规定屏幕宽度为20rem，同样在iPhone6中，屏幕宽度为375px，即20rem=375px，所以在iPhone6中1rem=18.75px。

以常规设备为例，这些尺寸换算如表2-1所示。

表2-1 rpx、rem与px换算关系示例

设备	rpx 换算 px (屏幕宽度 /750)	px 换算 rpx (750/ 屏幕宽度)	rem 换算 px (屏幕宽度 /20)	px 换算 rem (20/ 屏幕宽度)
iPhone5	1rpx=0.42px	1px=2.34rpx	1rem=15.75px	1px=0.06349..px
iPhone6	1rpx=0.5px	1px=2rpx	1rem=18.75px	1px=0.053px
iPhone6 Plus	1rpx=0.552px	1px=1.81rpx	1rem=20.7px	1px=0.04830..px

在设计界面时，如果要想实现尺寸自适应，设计师可以用iPhone6作为视觉标准。由于rpx和rem最终要被换算为px，在某些情况下可能会存在除不尽的情况，会导致界面中产生毛刺，这种情况大家要多留意、多测试以尽量避免这种情况。

2.选择器

CSS选择器用于选择需要添加样式的元素，WXSS实现了CSS的部分选择器，使用规则和CSS一样，熟悉CSS的同学会很快上手，参见表

2-2。

表2-2 选择器样例

选择器	样例	样例描述
.class	.myClassName	选择所有拥有 class="myClassName" 的组件
#id	#myIdName	选择拥有 id="myIdName" 的组件
element	view	选择所有 view 组件
element, element	view, checkbox	选择所有文档的 view 组件和所有 checkbox 组件
::after	view::after	在 view 组件后边插入内容
::before	view:before	在 view 组件前面插入内容

WXSS和CSS代码结构一样，一段样式前面是选择器，后面是以大括号括起来的样式组合，每个样式以分号结束，如下所示：

```
选择器 { 样式1; 样式2; }
```

选择器使用规则和CSS也是一致的，如下所示：

```
/* 选择所有class含有myClass的组件，并设置边框 */
.myClass { border : solid 1px #000; }

/* 选择所有view组件且class含有myClass的组件，并设置边框 */
view.myClass { border : solid 1px #000; }

/* 选择所有view组件中子节点class含有myClass的组件，并设置边框 */
view .myClass { border : solid 1px #000; }

/*
  选所有class含有myContent组件中所有checkbox组件和radiobox组件，并设置它们的边框
*/
.myContent checkbox,
.myContent radiobox { boder : solid 1px #000; }

/* 选择所有view组件且class含有myClass的组件，在其后面插入新内容，内容为new content*/
view.myClass::after { content : 'new content' }
```

3.内联样式

同HTML一样，样式除了写在WXSS文件中，也可以通过设置style、class属性控制样式，一般静态样式可以统一写到class中，style样式会在运行时解析，如非特别需要，尽量避免将静态样式写入style，以免影响渲染速度，例如：

```
<!-- 通过style动态设置样式 -->
<view style="border:solid 1px #000;background-color:{{color}}"></view>

<!-- 通过class选择器设置样式 -->
<view class="myClassName1 myClassName"></view>
```

4.样式导入

通常在项目中为了便于管理会将WXSS按职责拆分为多个文件，这时便需要@import语句在当前WXSS文件中导入其他WXSS文件，@import后写入需要导入WXSS文件的相对路径，用“；”表示语句结束，例如：

```
.common-view { border : solid 1px #000; }

@import "common.wxss";
.page-container { padding : 10px; }
```

至此，小程序框架页面相关的4个文件已介绍完成，大家对每个文件的功能、内容应该都有了一定了解，在本章最后一节中，我们将探讨小程序的模块化。

2.5 模块化

小程序逻辑层语言是JavaScript，而JavaScript作为脚本语言在设计初期仅是为了实现简单的页面交互，由Brendan Eich在1995年花了不到十天时间发明出来，语言本身缺失了很多用于支撑大型项目的设计，而现在前端业务逻辑越来越复杂，代码也越来越多，很多问题就暴露出来。模块化主要解决JavaScript中命名冲突和文件依赖这两个问题，现在模块化在前端中使用比较广泛，如Nodejs、Requirejs、Seajs、Webpack等，它们大部分都遵循或者接近CommonJS规范，甚至ES6也针对模块化提出了自己的规范。目前前端模块化没有一个统一的解决方案，在不同环境、不同框架中的实现都不一样，本节将重点讨论小程序的模块化规范。

2.5.1 模块化简介

最早前端JavaScript代码量不大，统一放在一个文件内，如下面一段代码：

```
var name = 'weixin',
    age = 12;

function getName() {
    // 实现代码
}
function getAge() {
    // 实现代码
}
```

后来前端代码越来越多，为了便于管理和工作拆分，我们不得不把代码拆分为多个文件，这时将上述代码封装到`user.js`文件中，需要用时引入页面（或打包到一个文件）就行。初期团队成员少，一切都运行正常，直到团队越来越大，开始有人抱怨：我想定义一个`name`变量但`user.js`中已经存在，我不得不定义为`myName`；为什么我在自己代码中定了`getAge`方法就导致别人代码出问题了呢？通过这种文件拆分的工作我们只是对代码做了物理上的分离，能初步实现多人开发和简单的代码管理，但并没有真正做到作用域的隔离，由于不知道其他文件内已存在的变量名，甚至让全局冲突问题变得更容易、更严重。

再后来为了避免这种全局冲突，大家决定参考Java的方式，引入命名空间和闭包来解决变量冲突问题。于是`user.js`里的代码变成了如下

这样:

```
( function() {  
  myProject = myProject || {}; // 定义全局命名空间  
  myProject.user = {};  
  myProject.user.name = 'weixin';  
  
  var age = 12; // 闭包内变量，外部不能访问  
  
  myProject.user.getName = function() {  
    // 实现代码  
  }  
  myProject.user.getAge = function() {  
    // 实现代码  
  }  
} )();
```

这样别的同事可以通过`myProject.user`获取`name`，调用`getName`和`getAge`方法，通过命名空间，的确能缓解大部分冲突，但是为此我们不得不记住很长一串命名空间，同时当我使用`user`这个空间后，别人就不能使用，这也不能完美地解决问题。同时更可怕的是如果`user.js`依赖另外一个`utils.js`，别的同事必须通过阅读`user.js`源码搞懂这层依赖关系，按顺序引入`utils.js`、`user.js`，直接引入`user.js`将会导致他代码出错，如果`utils`还依赖别的资源他还得必须搞懂相关的所有依赖，而他仅仅是想调用我的`getName`方法，这对调用的同事来说无疑是个噩梦。这时我们需要一种新的组织方式，于是诞生了模块化：

- 模块是一段JavaScript代码，具有统一的基本书写格式。

- 模块之间通过基本交互规则，能彼此引用，协同工作。

目前模块化的规范不统一，大致可分为CommonJS和ES6两种规范，大家有兴趣可以参考网上相关资料，小程序模块化机制比较接近CommonJS规范，无论哪种规范，学习起来都十分简单。

2.5.2 文件作用域

小程序中一个JavaScript文件就是一个模块，在这个文件中声明的变量和函数只在该文件中有效，不同文件中的相同变量名和函数名是不会互相影响的。模块中可以调用一些全局的方法，如下例中通过调用getApp（）获取小程序实例：

```
App( {
  myGlobalData : { /* 定义全局属性 */
    name : 'weixin'
  }
} );

var myPrivatyData = "value1"; /* myPrivatyData只能在a.js中使用 */
var appData = getApp();
appData.myGlobalData.name += ' app';

var myPrivatyData = "value2"; /* myPrivatyData不会和a.js中同名变量冲突 */
var appData = getApp();
/* 当a.js在b.js前执行后，这里会输出"weixin app value2" */
console.log( appData.myGlobalData.name + ' ' + myPrivatyData );
```

2.5.3 模块的使用

模块接口的暴露和引入十分简单：

- 通过`exports`暴露接口。

- 通过`require (path)`引入依赖，`path`是需要引入的模块文件的相对路径。

示例代码如下：

```
var privateData = 'weixin';

function run( who ) {
  console.log( who + ' run' );
}
function walk( who ) {
  console.log( who + ' walk' );
}
module.exports.run = run;
exports.walk = walk;

/**
 也可以这样
  module.exports = {
    run : run,
    walk : walk
  };
*/

var otherMod = require( 'mod.js' ); /* */

Page( {
  onShow : function() {
    /* 这里会打印出somebody run */
    otherMod.run( 'somebody' );
    /* 这里会打印出somebody walk */
    otherMod.walk( 'somebody' );
  }
} );
```

需要注意的是：

·`exports`是`module.exports`的一个引用，因此在模块里面随意更改`exports`的指向会造成未知的错误。所以我们更推荐开发者采用`module.exports`来暴露模块接口，除非你已经很清晰地知道这两者的关系。

·小程序目前不支持直接引入`node_modules`，开发者需要使用`node_modules`时建议拷贝出相关代码到小程序目录中。

通过模块化我们能实现代码真正的隔离，可以多人并行开发，降低大型项目管理难度，这对前端工程化具有很大促进作用。

2.5.4 其他

1.JavaScript运行环境

微信小程序逻辑代码运行在三端：iOS、Android和用于调试的开发者工具，这三端是各自不同的三个解析引擎：

- 在iOS上，小程序的JavaScript代码是运行在JavaScriptCore中。
- 在Android上，小程序的JavaScript代码是通过X5内核来解析。
- 在开发工具上，小程序的JavaScript代码是运行在nwjs（chrome内核）中。

虽然尽管三端的环境十分相似，但是至少在目前对一些语法、特性的支持还是有一些区别，在开发过程中要尽可能地在三端进行测试。

2.ES6语法以及API支持

在小程序中，开发者可以使用ES6语法进行编码，在0.10.101000以及之后版本的开发工具中，会默认使用babel将开发者ES6代码转换为三端都能很好支持的ES5的代码，帮助开发者解决环境不同所带来

的开发问题。如果没有使用**ES6**语法，开发者可以在项目设置中关闭这个功能。

转化过程中需要注意：

- 这种转换只会帮助开发者处理语法上问题，新的**ES6**的**API**，例如**Promise**等需要开发者自行引入**Polyfill**或者别的类库。

- 为了提高代码质量，在开启**ES6**转换功能的情况下，默认启用**JavaScript**严格模式，请参考“**use strict**”。

2.6 小结

小程序框架是小程序开发的核心，本章重点讲解了框架的大致原理、规范，剖析了所涉及的每个文件，小程序这种基于数据的编码方式和前端基于**DOM**的编码方式有很大不同，大家要着重了解熟悉，了解小程序运行原理后，后续学习组件和**API**将会非常简单。

第3章 布局

WXSS实现了CSS布局相关的大部分规范，但在一些细节上有差异，甚至同样的语法在小程序调试工具和微信中的表现也存在差异。本章主要讲述CSS布局相关的一些基本知识，包括经典的盒子模型、浮动定位、绝对定位以及近几年提出的Flex布局。这些基本知识在WXSS也是通用的。对于其他一些特性，开发者可在开发过程中尝试，如果大家想对CSS有更深入的了解可以参考网络资料或《CSS权威指南》。这里再次提醒大家，在代码编写过程中一定要开启开发者工具中的这个功能：“开启上传代码时样式文件自动补全”，否则在学习本章Flex布局时会存在不同终端兼容性问题。

3.1 基本知识

3.1.1 盒子模型

盒子模型是CSS布局的基础，CSS假定每个元素都会生成一个或多个矩形框，每个元素框中心都有一个内容区（**content**），这个内容区周围有内边距（**padding**）、边框（**border**）和外边距（**margin**），这些项默认宽度为0，这个矩形框就是常说的盒子模型，如图3-1所示。

简单来说，HTML中每一个元素就是一个盒子，同理，在小程序中每一个组件就是个盒子，元素的宽度、高度就是内容区域宽度、高度，不包含内边距、边框和外边距，我们可以通过元素**width**、**height**、**padding**、**border**、**margin**属性控制盒子样式。盒子模型根据浏览器具体实现可分为W3C的标准盒子模型和IE盒子模型，这两种盒子模型在宽度和高度的计算上不一致，IE盒子模型的宽度和高度是包含内边距和边框的，我们这里讲述的主要是W3C的盒子模型，WXSS完全遵守W3C盒子模型规范。

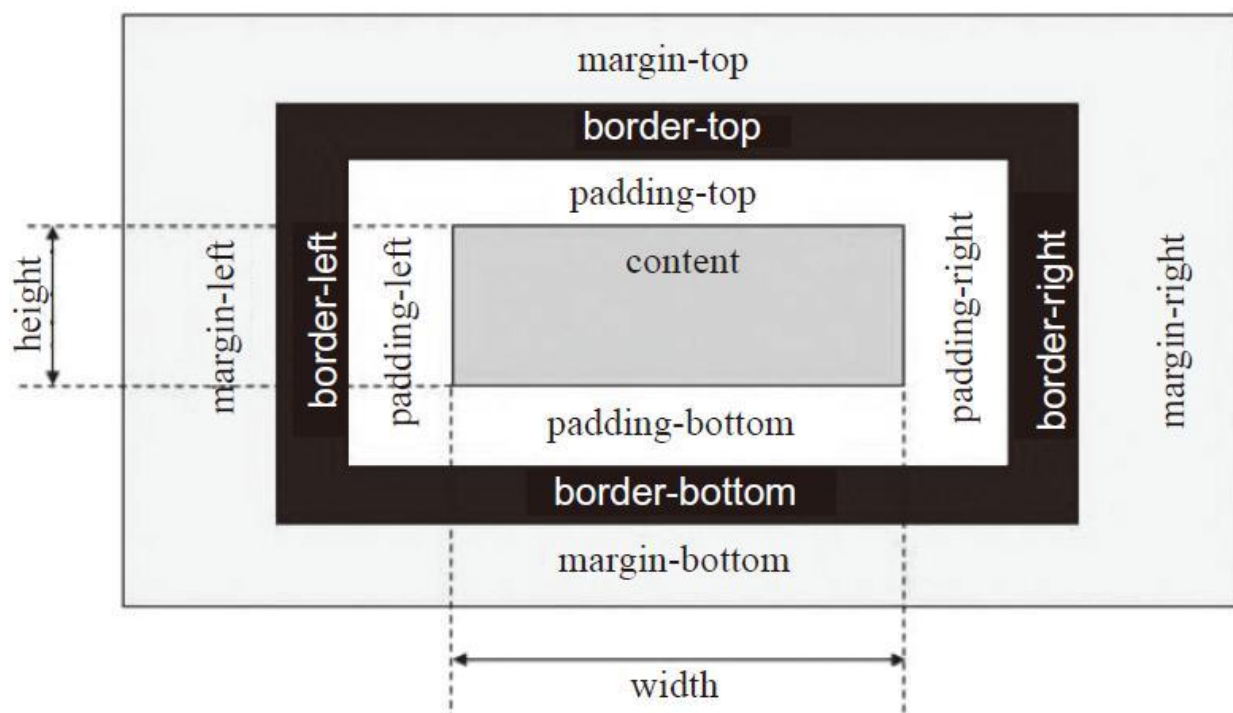


图3-1 盒子模型

CSS中的布局都是基于盒子模型，不同类型元素对盒子模型的处理也是不同的，块级元素的处理就和行内元素不同，浮动元素和定位元素的处理也是不相同的，接下来我们逐一讨论这些差异。

3.1.2 块级元素

元素按显示方式主要可以分为块级元素和行内元素，元素的显示方式是由`display`属性控制的，块级元素会默认占一行高度，一般一行内只有一个块级元素（浮动后除外），当再添加新的块级元素时，新元素会自动换行显示。块级元素一般作为容器出现，用于组织结构。一些元素默认就是块级元素，如小程序中的`<view/>`组件，而一些则默认是行内元素，我们可以通过修改元素`display`属性为`block`，将一个元素强制设置为块级元素。一个块级元素的元素框与其父元素的`width`相同，块级元素的

$\text{width} + \text{marginLeft} + \text{marginRight} + \text{paddingLeft} + \text{paddingRight}$ 刚好等于父级元素内容区宽度，显示时默认撑满父元素内容区。块级元素高度由其子元素决定，父级元素高度会随内容元素变化而变化。块级元素特点总结如下：

- 总是在新行上开始。

- 宽度默认为

$\text{width} + \text{marginLeft} + \text{marginRight} + \text{paddingLeft} + \text{paddingRight}$ 刚好等于父级元素内容区宽度，除非设定一个新宽度，这里需要注意，当设置块级元素宽度为100%时，如果当前块级元素存在`padding`、`margin`会导致块级元素溢出父元素。

- 盒子模型高度默认由内容决定。
- 盒子模型中高度、宽度及外边距和内边距都可控制。
- 可以容纳行内元素和其他块级元素。

示例：

<view/>组件默认是块级元素，下面我们使用<view/>为大家演示块级元素的特性，如图3-2所示。

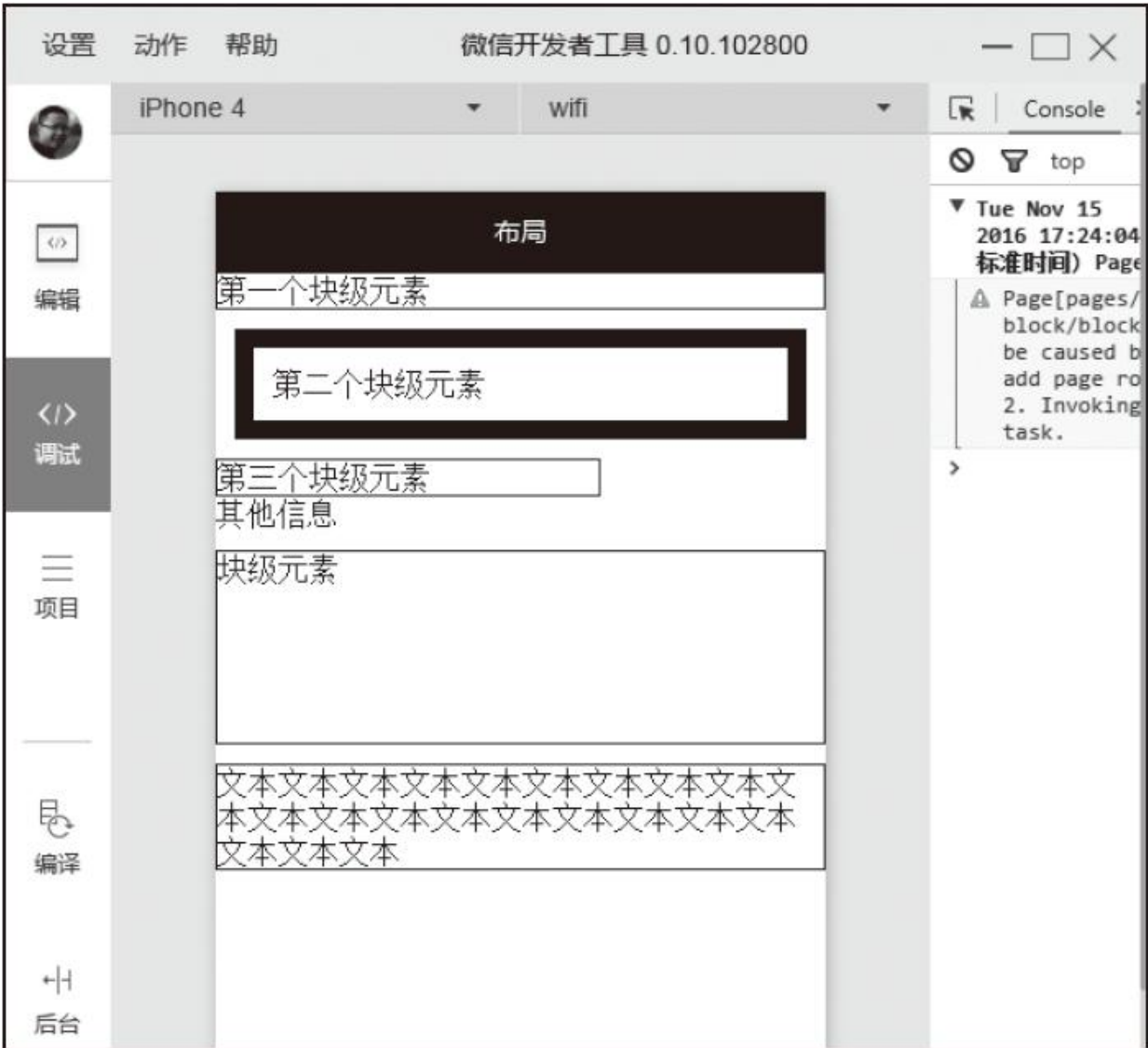


图3-2 块级元素示例

代码如下：

```
<!-- 每个块级元素占领一行 -->
<view style="border:solid 1px;">第一个块级元素</view>

<!-- 默认情况下块级元素的元素框和父级元素的width相同，刚好撑满内容区 -->
<view style="border:solid 10px; margin : 10px; padding :10px;">第二个块级元素
</view>

<!-- 即使宽度不足，仍会占领一行让其余元素换行 -->
<view style="border:solid 1px; width : 200px;">第三个块级元素</view>
其他信息
```


3.1.3 行内元素

除了块级元素，最常见的就是行内元素了，通过设置`display`属性为`inline`可以将一个元素设置为行内元素，小程序中`<text/>`就是一个行内组件。行内元素没有块级元素那么简单直接，块级元素只是生成框，通常不允许其他内容与这些框并存，行内元素特点总结如下：

- 和其他非块级元素都在一行上。
- 盒子模型中高度、宽度、上下`margin`、上下`padding`设置均无效，只能设置左右`margin`和左右`padding`。
- 宽度就是文字或图片的宽度，不可改变。
- 行内元素宽度、高度都不能直接设置。
- 行内元素只能容纳文本或其他行内元素，在行内元素中放置块级元素会引起不必要的混乱。

如图3-3中的示例，大家可以对比本例中块级元素和行内元素的区别，我们设置了行内元素的`margin`，布局时上下`margin`都被忽略了。本例中我们将上下`padding`设置为0，大家可以尝试设置为其他值，这时会发现上下`padding`会生效，但是不会影响布局，本例中行内元素换行是因为上面的块级元素强制占位一行。

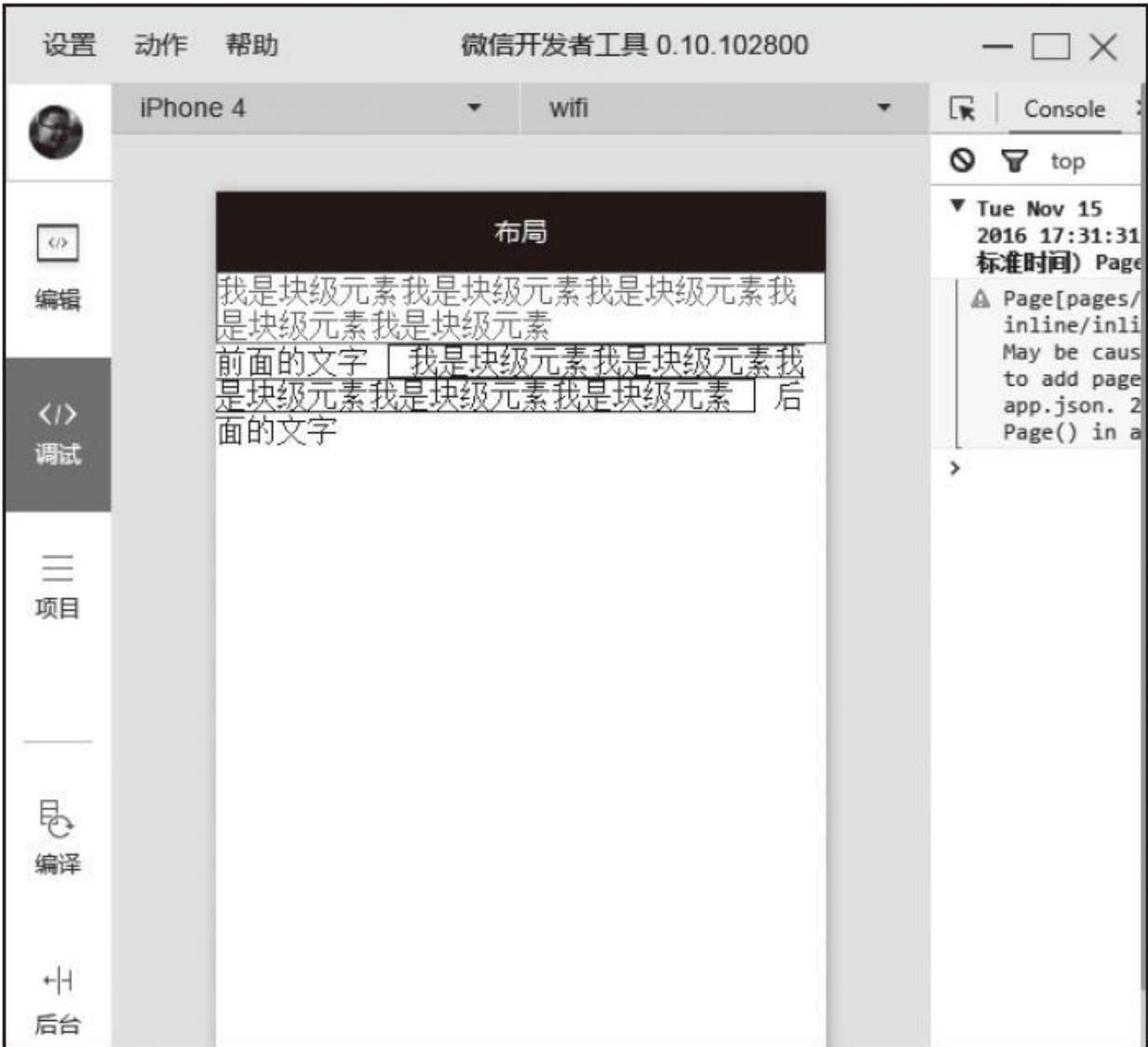


图3-3 行内元素示例

本例的代码如下：

```
<!-- 块级元素 -->
<view style="border:solid 1px #999; color : #999;">我是块级元素我是块级元素我是块级元素
我是块级元素我是块级元素</view>

<!-- 通过修改display属性的行内元素 -->
前面的文字<view style="border:solid 1px; margin:10px; padding : 0 10px; display:
inline;">我是块级元素我是块级元素我是块级元素我是块级元素我是块级元素</view>后面的文字
```

3.1.4 行内块元素

行内块元素是块级元素和行内元素的混合物，当`display`属性为`inline-block`时，元素就被设置为一个行内块元素，行内块元素可以设置宽、高、内边距和外边距，可以简单认为行内块元素是把块级元素以行的形式展现，保留了块级元素对宽、高、内边距、外边距的设置，它就像一张图一样放在一个文本行中，如图3-4所示。

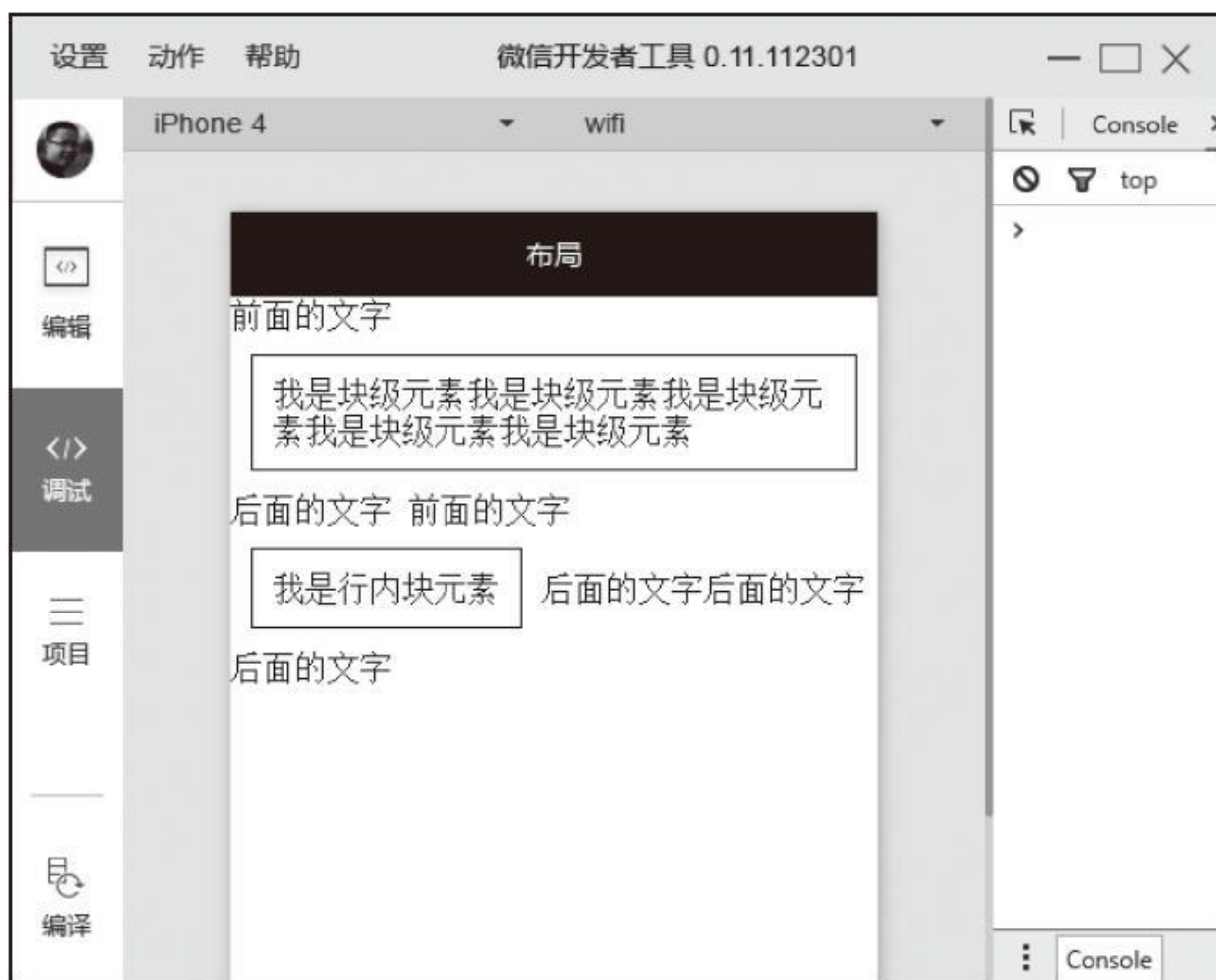


图3-4 行内块元素示例

代码如下：

```
<!-- 行内块元素宽度撑满父级宽度情况 -->
前面的文字<view style="border:solid 1px; margin:10px; padding : 10px;
display:inline-block;">我是块级元素我是块级元素我是块级元素我是块级元素我是块级元素</view>后
面的文字

<!-- 行内块元素宽度不足父级宽度情况 -->
前面的文字<view style="border:solid 1px; margin: 10px; padding : 10px; display:
inline-block;">我是行内块元素</view>后面的文字后面的文字后面的文字
```

3.2 浮动和定位

了解基本盒子模型后，本小节开始讲解定位相关的内容，定位的基本思想很简单，它允许你定义元素框相对于其正常位置应该出现在哪，或者相对于父元素、另一个元素甚至浏览器窗口本身的位置。浮动和定位是我们常用的布局方案，**WXSS**也支持**Flex**布局方案，接下来我们将对这三种布局方案一一讲解。

3.2.1 浮动

浮动不完全是定位，同时它也不是正常流布局，通过设置float属性，浮动的框可以向左或者向右移动，直到其外边缘碰到包含框或另一个浮动框的边框为止。由于浮动框不在文档的普通流中，文档的普通流中的会表现的浮动框不存在一样，其他内容会环绕过去，如图3-5所示。

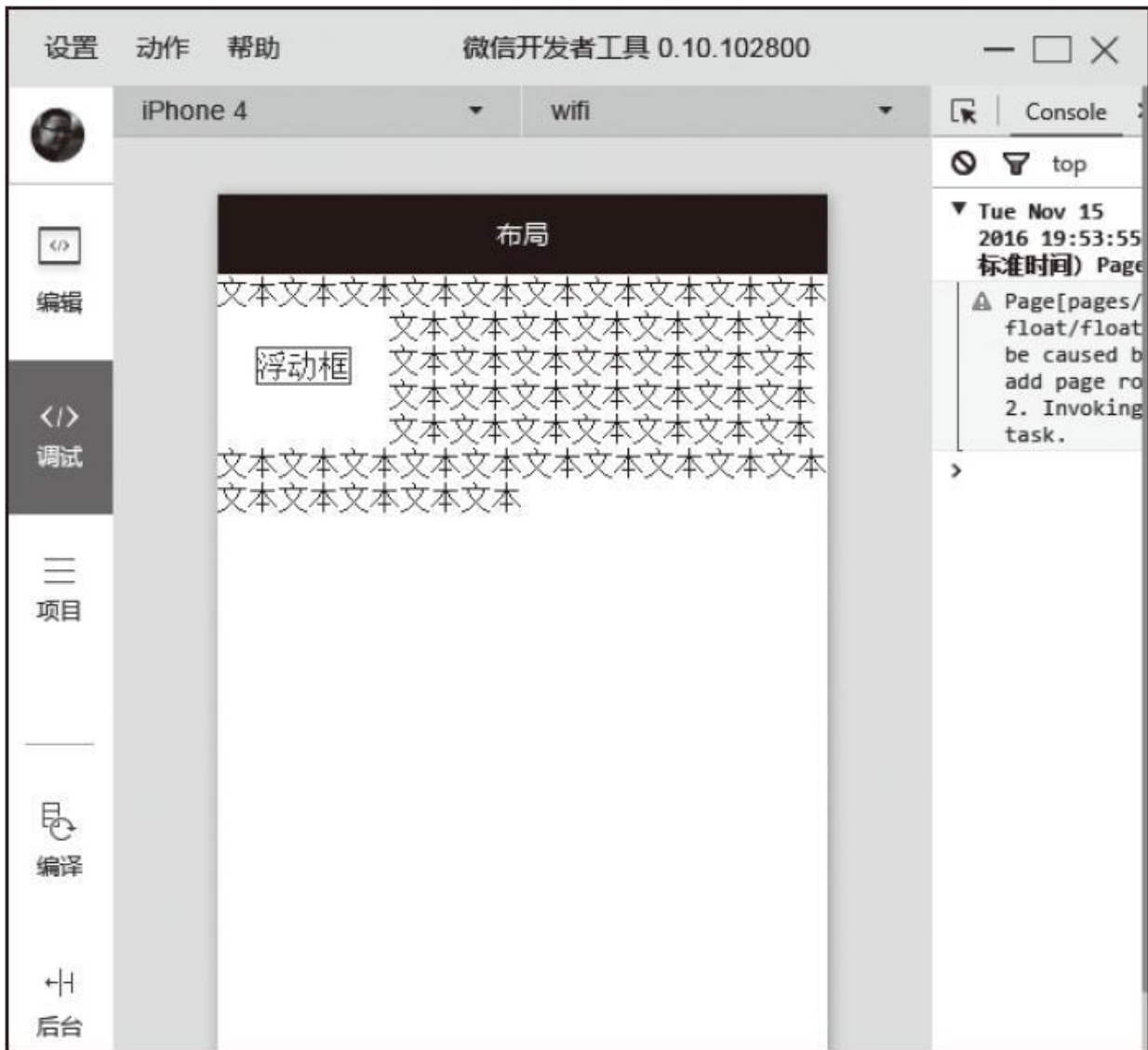


图3-5 浮动示例

代码如下:

```
<view>
  文本文本文本文本文本文本文本文本文本文本文本文本
<view
style="display:block;float:left;border:solid 1px;margin : 20px;">浮动框</view>文本文
本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本
文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本
</view>
```

上例中浮动区域在它当前的位置往左浮动，直至父元素内容框，其他文本都环绕而过。由于元素浮动时不在普通流中，这会导致父级元素忽略浮动元素高度，形成坍塌，如图3-6所示。

代码如下：

```
<!-- 父级元素高度只会包含第一元素 忽略浮动元素 -->
<view style="border:solid 1px;">
  <view>其他元素</view>
  <view style="float:left;">浮动框</view>
</view>
```

本例中父级元素的边框并没有包裹浮动框，虽然这是浮动的一个特

本例中父级元素的边框并没有包裹浮动框，虽然这是浮动的一个特性，并不是一个bug，但在某些情况下我们仍然希望在使用浮动的同时，父级元素的高度能包裹浮动元素，这时我们就需要了解和浮动的另一个属性：**clear**（清除）。当设置元素**clear**时，可以确保当前元素的左边、右边或左右两边同时不能出现浮动的元素，如图3-7所示。

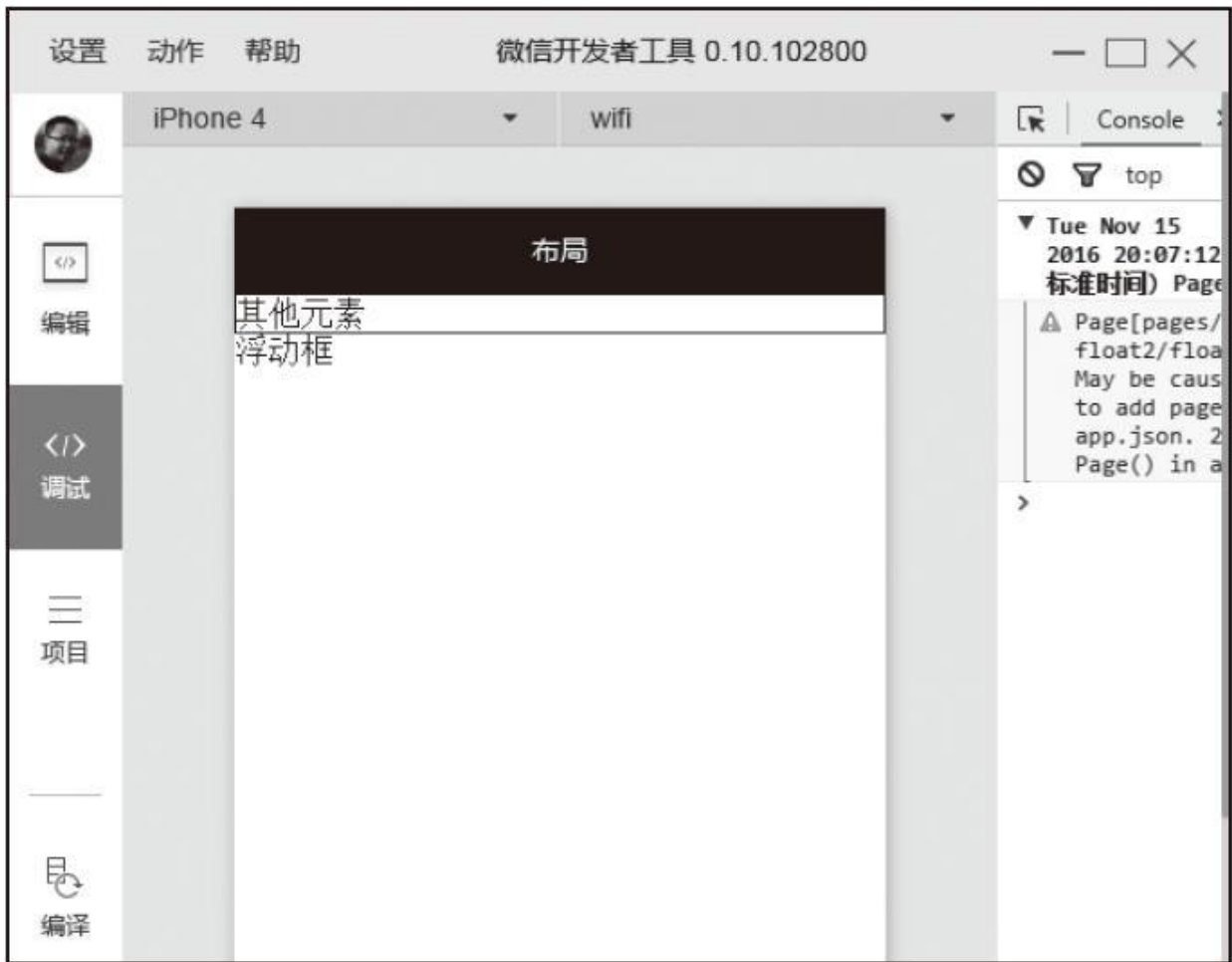


图3-6 浮动高度问题示例

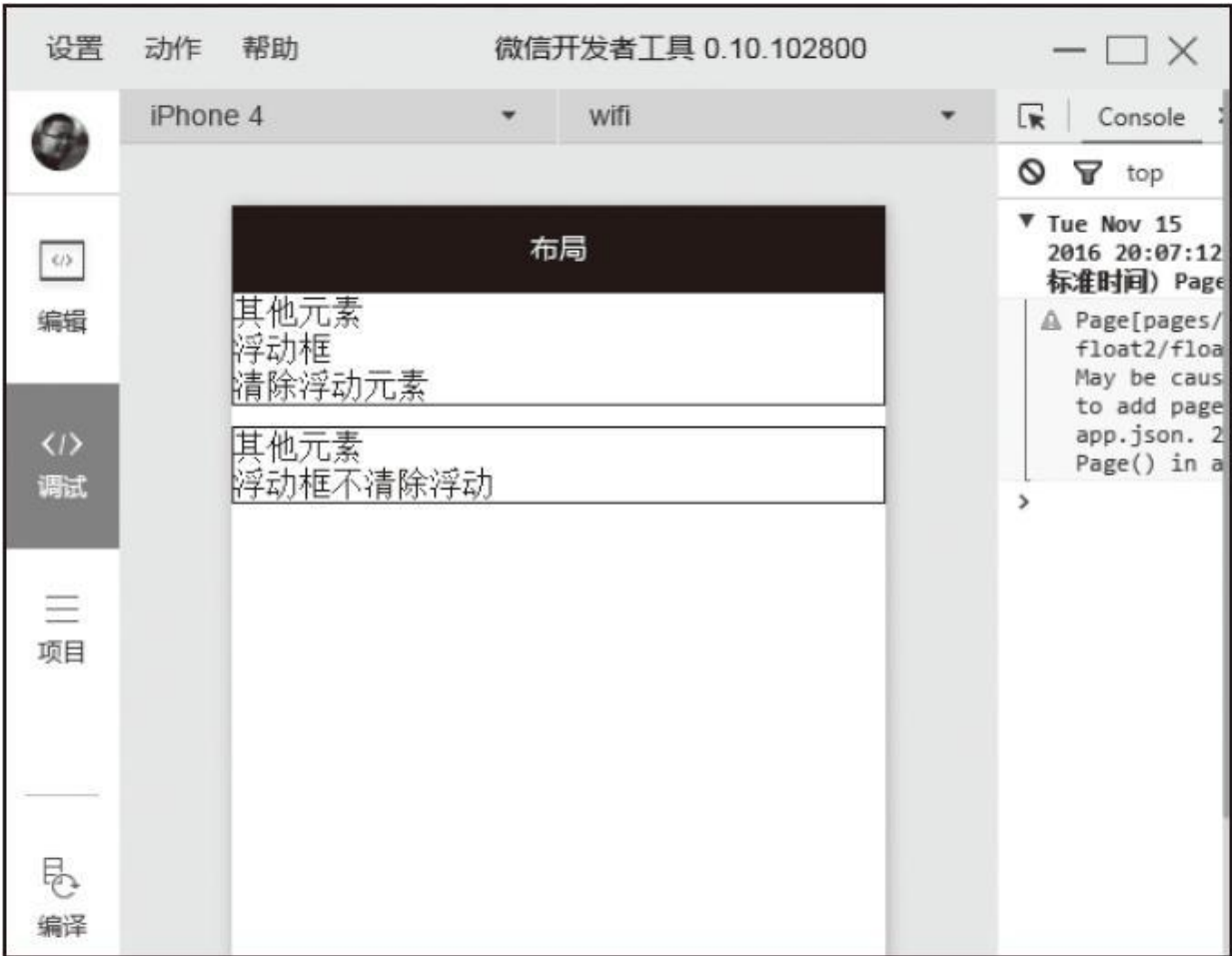


图3-7 清除浮动示例

代码如下：

```
<!-- 父元素会包含清除浮动元素 -->
<view style="border:solid 1px;">
  <view>其他元素</view>
  <view style="float:left;">浮动框</view>
  <!-- 设置当前元素左边不能出现浮动元素 -->
  <view style="clear:left;">清除浮动元素</view>
</view>

<view style="border:solid 1px; margin-top:10px;">
  <view>其他元素</view>
  <view style="float:left;">浮动框</view>
  <view>不清除浮动</view>
</view>
```

在上例中有个特别有意思的现象，父元素虽然会忽略浮动元素（如浮动高度示例中产生的坍塌），但是不会忽略其他元素（包括清除浮动的元素），而清除浮动的元素总在浮动元素下方，所以在显示时视觉上父元素就把所有元素都包含进去了，如上例中无论非浮动元素在哪里，父元素边框都包含了非浮动元素。利用这个特性，如果把上例中清除浮动的高度置为0使其看不见，这时父元素仍然会包裹它，这样就能防止浮动元素父元素高度坍塌，网上利用`after`伪属性清除浮动就是这个原理。这里我们对比使用元素和`after`伪属性2种实现方案，如图3-8所示。

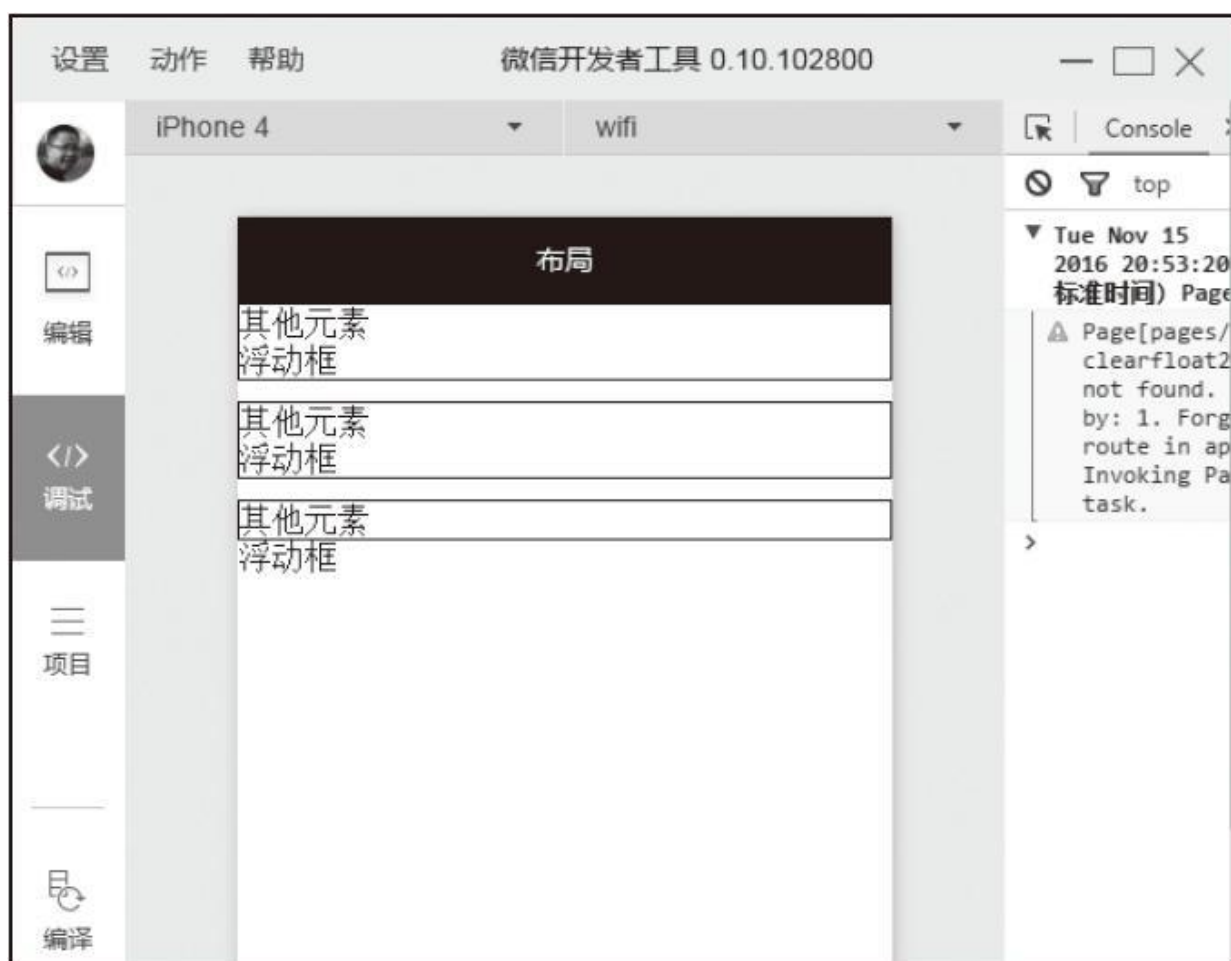


图3-8 清除浮动应用示例

WXML文件的代码如下:

```
<!-- 添加高度为0的元素清除浮动 -->
<view style="border:solid 1px;">
  <view>其他元素</view>
  <view style="float:left;">浮动框</view>
  <view style="clear:both;height:0;"></view>
</view>

<!-- 利用伪属性在后面插入一个元素清除浮动 -->
<view style="border:solid 1px; margin-top:10px;" class="clearfix">
  <view>其他元素</view>
  <view style="float:left;">浮动框</view>
</view>

<!-- 不清除浮动对比 -->
<view style="border:solid 1px; margin-top:10px;">
  <view>其他元素</view>
  <view style="float:left;">浮动框</view>
</view>
```

.WXSS文件的代码如下:

```
/* 一定要设置content, 否则元素不会显示 */
.clearfix:after { display:block; height : 0; clear : both; content : '' }
```

在实际项目中, 为了复用性和便捷性, 我们通常使用.clearfix类清除浮动。

3.2.2 定位

元素的定位由`position`属性控制，`position`有4种不同类型的定位，会影响元素框生成的方法：

- static**：元素框正常生成。块级元素生成一个矩形框，作为文档流的一部分，行内元素则会创建一个或多个行框，置于其父元素中，**static**是**position**的默认值。

- relative**：元素框偏移某个距离。元素仍保持其未定位前的形状，它原本所占的空间仍保留。

- absolute**：元素框从文档流中完全删除，并相对于其包含块定位，包含块可能是文档中的另一个元素或者是初始包含块。对于**absolute**来说，包含块是离当前元素最近的**position**为**absolute**或**relative**的父元素，如果父元素中没有任何**absolute**或**relative**布局的元素，那么包含块就是根元素。使用**position**布局后，元素原先在正常文档流中所占用的空间会关闭，就好像该元素原来不存在一样。元素定位后生成一个块级框，不论原来它在正常流中生成何种类型的框。

- fixed**：元素框的表现类似于将**position**设置为**absolute**，不过其包含块是视窗本身。

示例如图3-9所示。



图3-9 position属性示例

代码如下:

```
<!-- relative相对之前位置进行移动，原占有空间不会被关闭 -->
<view style="border:solid 1px;">
  文案文案<text style="position:relative; top : 10px; left :
```

```
10px;">relative</text>文案文案文案文案文案文案文案文案文案文案
</view>
```

```
<!-- absolute依赖于包含块，原占有空间会被关闭 -->
<view style="border:solid 1px; position:relative; height : 80px;">
  文案文案<text style="position:absolute;left : 10px; bottom :
10px;">absolute</text>文案文案文案文案文案文案文案文案文案文案
</view>
```

```
<!-- 没有找到最近的absolute或relative元素会直接认为根元素是包含块，原占有空间会关闭 -->
<view style="border:solid 1px;">
  文案文案<text style="position:absolute;left : 10px; bottom : 10px;">absolute不设置包含块</text>文案文案文案文案文案文案文案文案文案文案
</view>
```

```
<!-- fixed直接认为视窗本身为包含块，原占有空间会关闭 -->
<view style="border:solid 1px;">
  文案文案<text style="position:fixed;right : 10px; bottom : 30px;border:solid
1px;">fixed</text>文案文案文案文案文案文案文案文案文案文案
</view>
```

3.3 Flex布局

浮动和定位是基于盒子模型的传统布局解决方案，它在处理一些特殊布局时非常不方便，比如垂直居中，2009年W3C提出了一种新的方案Flex布局，该布局可以简单快速地完成各种伸缩性的设计。Flex是Flexible Box的缩写，即为弹性盒子布局，可以为传统盒子模型带来更大的灵活性，目前主流浏览器都支持这种布局，小程序WXSS也对其进行了实现，项目中可以随意使用。

3.3.1 基本概念

Flex布局主要由容器和项目构成，采用Flex布局的元素，称为Flex容器（flex container），它的所有直接子元素自动成为容器成员，称为Flex项目（flex item）。可以设置`display: flex`或`display: inline-flex`将任何一个元素指定为Flex布局，如图3-10所示，容器默认存在两根轴，水平的主轴（main axis）和垂直的交叉轴（cross axis），主轴开始的位置（及主轴与边框的交叉点）叫main start，结束的位置叫main end，main start和main end和主轴的方向有关；交叉轴开始的位置叫cross start，结束的位置叫cross end，cross start和cross end和交叉轴方向有关。项目默认沿主轴从主轴开始的位置到主轴结束的位置进行排列，项目在主轴上占据的空间叫main size，在交叉轴上占据的空间叫cross size。

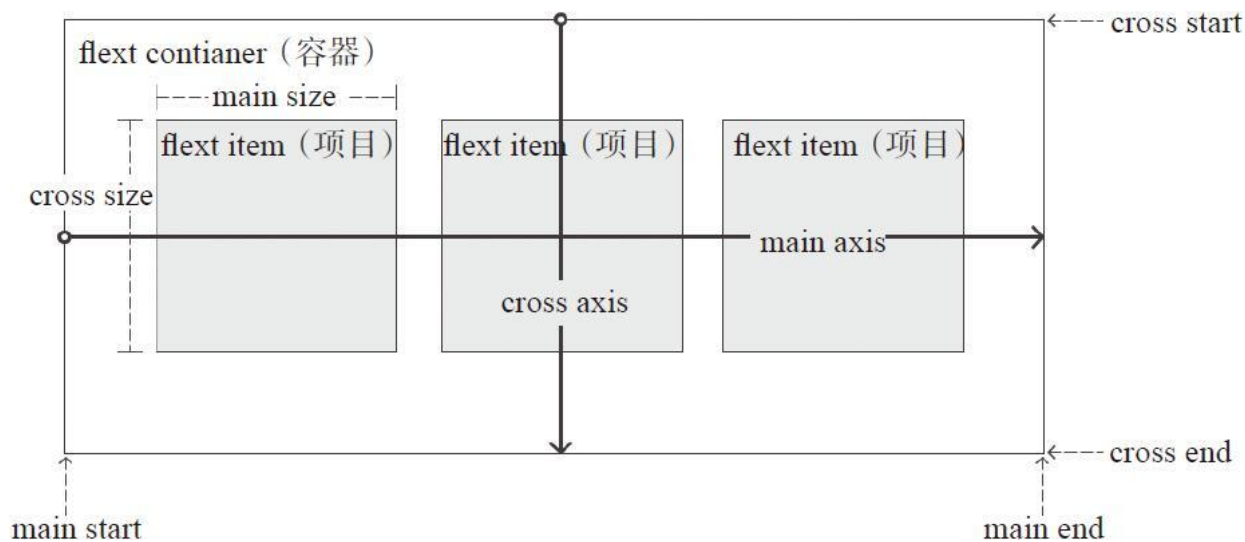


图3-10 Flex布局模型

3.3.2 容器属性

容器支持的属性有：

·**display**：通过设置**display**属性，指定元素是否为**Flex**布局。

·**flex-direction**：指定主轴方向，决定了项目的排列方式。

·**flex-wrap**：排列换行设置。

·**flex-flow**：**flex-direction**和**flex-wrap**的简写形式。

·**justify-content**：定义项目在主轴上的对齐方式。

·**align-items**：定义项目在交叉轴上的对齐方式。

·**align-content**：定义多根轴线的对齐方式，如果只有一根轴线，该属性不起作用。

1.display

display用来指定该元素是否为**Flex**布局，语法为：

```
.mybox{ display: flex | inline-flex; }
```

其属性值如下：

·**flex**: 用于产生块级Flex布局。

·**inline-flex**: 用于产生行内Flex布局，行内容器符合行内元素的特性，同时在容器内又符合Flex布局规范。

设置Flex布局以后，子元素的float、clear和vertical-align属性将会失效。

2.flex-direction

flex-direction用于指定主轴的方向，即项目排列的方向，语法为：

```
.mybox{ flex-direction : row | row-reverse | column | column-reverse; }
```

它有4个值（如图3-11）：

·**row**: 主轴为水平方向，起点在左端，默认值。

·**row-reverse**: 主轴为水平方向，起点在右端。

·**column**: 主轴为垂直方向，起点在上沿。

·**column-reverse**: 主轴为垂直方向，起点在下沿。

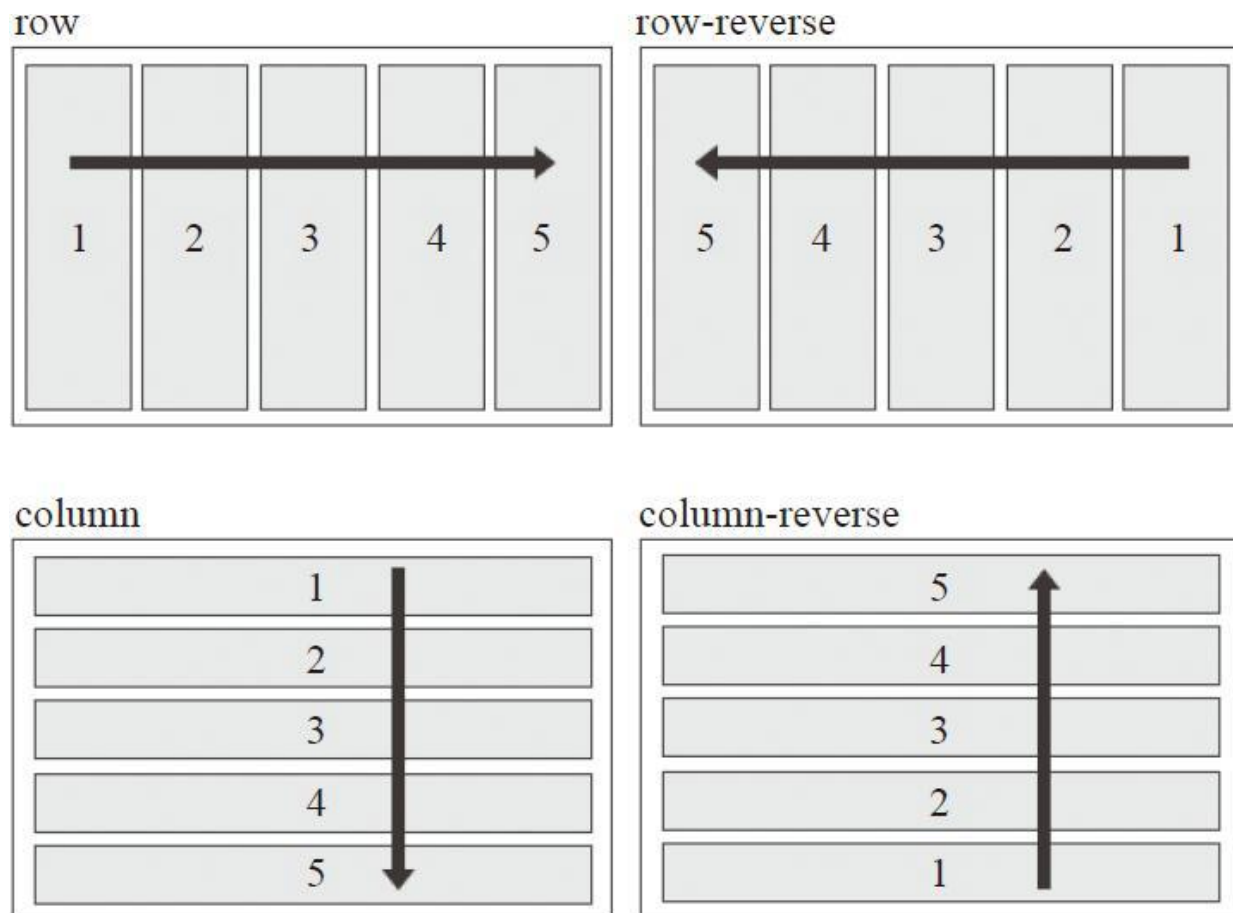


图3-11 flex-direction示例

3.flex-wrap

默认情况下，项目都排在一条线上，**flex-wrap**用来指定如果一条轴线排不下，该如何换行。

```
.mybox{ flex-wrap : nowrap | wrap | wrap-reverse; }
```

它有3个值（如图3-12所示）：

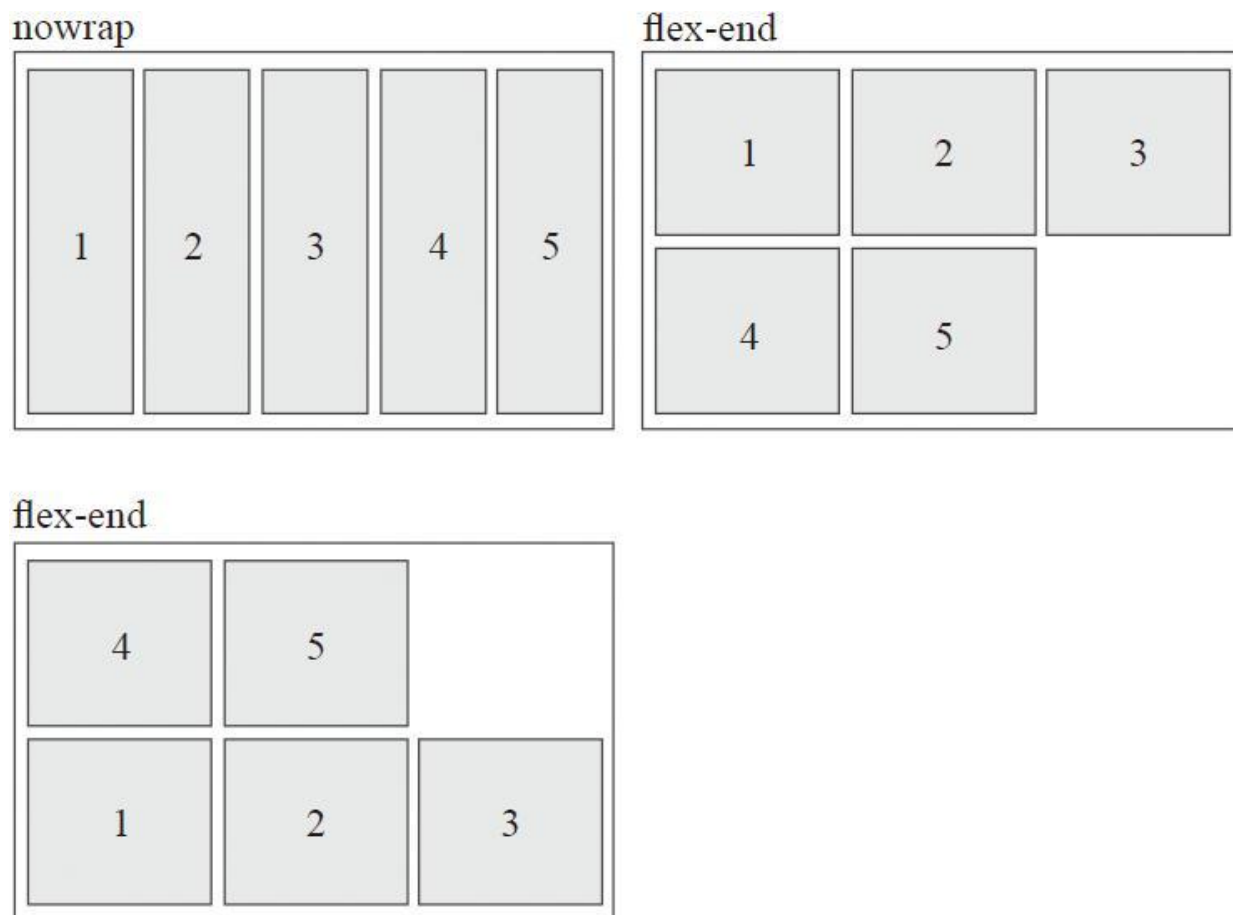


图3-12 flex-wrap示例

- nowrap**: 不换行，默认值。
- wrap**: 换行，第一行在上方。
- wrap-reverse**: 换行，第一行在下方。

当设置换行时，还需要设置**align-item**属性配合实现自动换行，并且**align-item**的值不能为“stretch”。

4.flex-flow

`flex-flow`是`flex-direction`和`flex-wrap`的简写形式，默认值为`row nowrap`。语法如下：

```
.mybox{ flex-flow : <flex-direction> || <flex-wrap>; }
```

示例代码如下：

```
.mybox{ flex-flow : row-reverse wrap; }  
.mybox{ flex-flow : wrap row-reverse; } /* 和上一句效果一样 */  
.mybox{ flex-flow : row-reverse; }  
.mybox{ flex-flow : wrap; }
```

5.justify-content

`justify-content`属性定义了项目在主轴上的对齐方式，语法如下：

```
.mybox{ justify-content : flex-start | flex-end | center | space-between | space-around; }
```

`justify-content`与主轴方向有关，包括以下属性（默认主轴从左到右，如图3-13所示）：

·`flex-start`：左对齐，默认值。

·`flex-end`：右对齐。

·`center`：居中。

·`space-between`：两端对齐，项目之间的间隔都相等。

·**space-around**: 每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

6.align-items

align-items指定项目在交叉轴上如何对齐，语法如下：

```
.mybox{ align-items : flex-start | flex-end | center | baseline | stretch; }
```

align-items与交叉轴方向有关，包括以下属性（假设交叉轴从上到下，如图3-14所示）：

·**flex-start**: 交叉轴的起点对齐。

·**flex-end**: 交叉轴的终点对齐。

·**center**: 交叉轴的中线对齐。

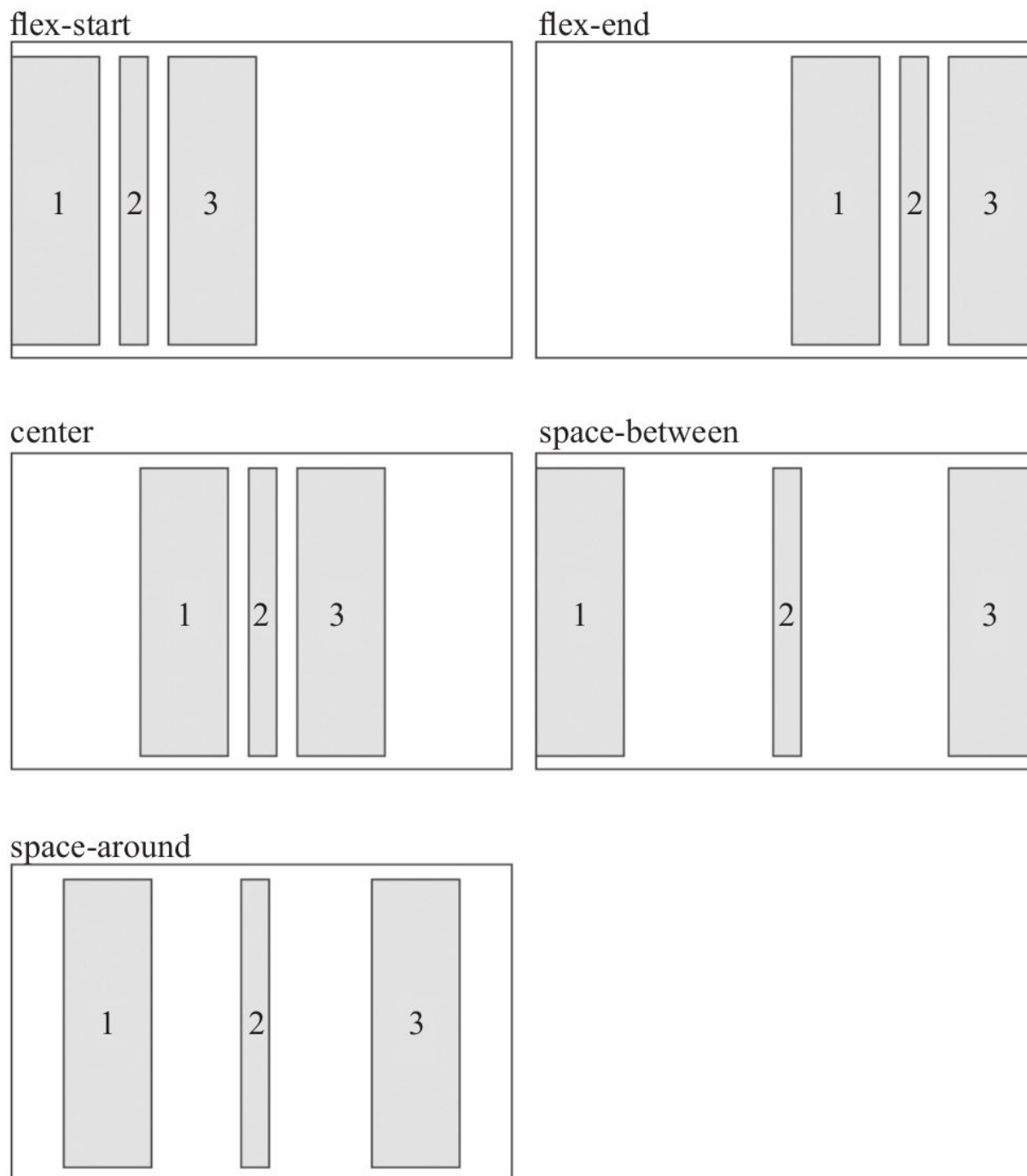


图3-13 justify-content示例

·baseline: 项目根据它们第一行文字的基线对齐。

·**stretch**: 如果项目未设置高度或设置为**auto**，项目将在交叉轴方向拉伸填充整个容器，默认值。

7.align-content

align-content用来定义项目多根轴线（出现换行后）在交叉轴上的对齐方式，如果项目只有一根轴线，该属性不起作用，语法如下：

```
.mybox{ align-content : flex-start | flex-end | center | space-between | space-around | stretch; }
```

其属性值如下（如图3-15）：

·**flex-start**: 与交叉轴的起点对齐。

·**flex-end**: 与交叉轴的终点对齐。

·**center**: 与交叉轴的中点对齐。

·**space-between**: 与交叉轴两端对齐，轴线之间的间隔平均分布。

·**space-around**: 每根轴线两侧的间隔都相等，轴线之间的间隔比轴线与边框间隔大一倍。

·**stretch**: 轴线占满整个交叉轴，每个项目会被拉伸直至填满交叉轴，默认值。

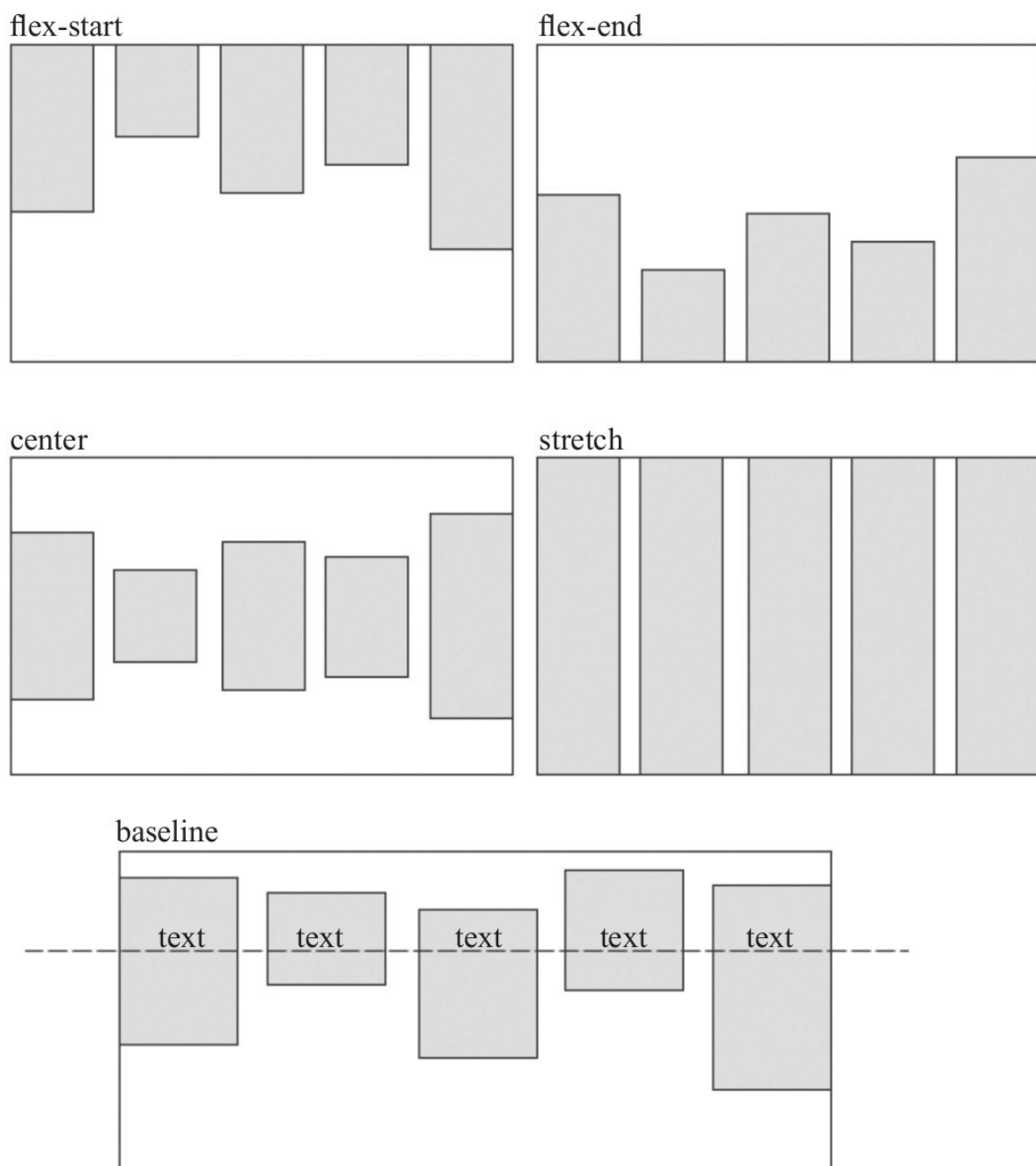


图3-14 align-items示例

3.3.3 项目属性

项目支持的属性有6个：

·**order**：定义项目的排序顺序。

·**flex-grow**：定义项目的放大比例。

·**flex-shrink**：定义项目的缩小比例。

·**flex-basis**：定义在分配多余空间之前，项目占据的主轴空间（main size）。

·**flex**：flex-grow、flex-shrink和flex-basis的简写。

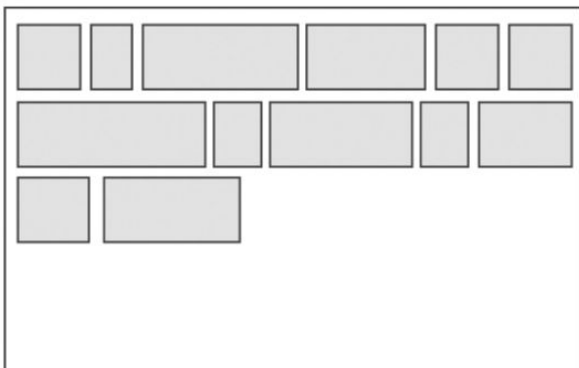
·**align-self**：用来设置单独的伸缩项目在交叉轴上的对齐方式，可覆盖默认的align-items属性。

1.order

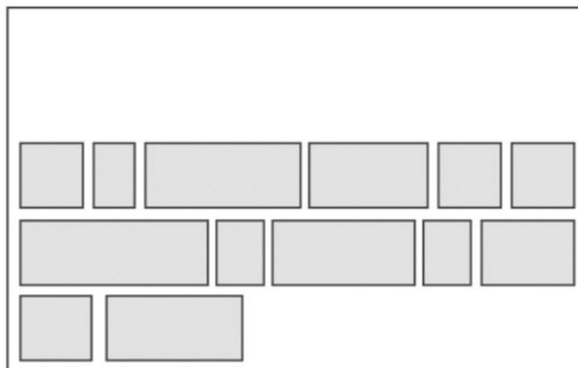
order属性定义项目的排列顺序。数值越小，排列越靠前（如图3-16），默认为0，语法如下：

```
.myitem{ order : <integer>; }
```

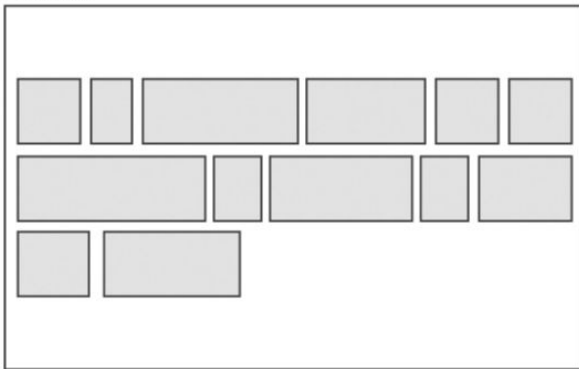
flex-start



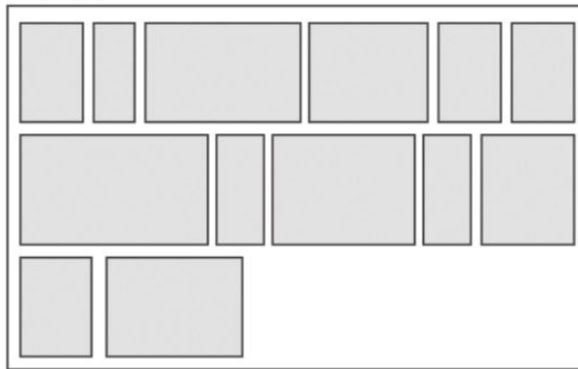
flex-end



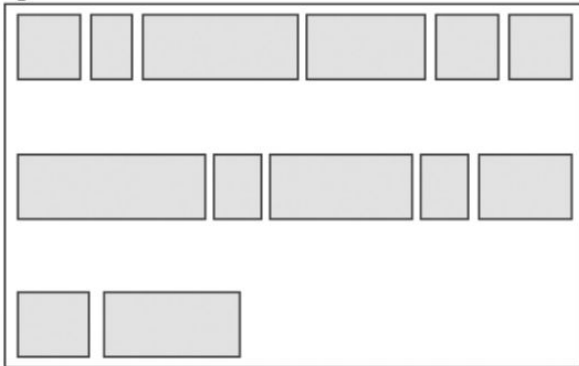
center



stretch



space-between



space-around

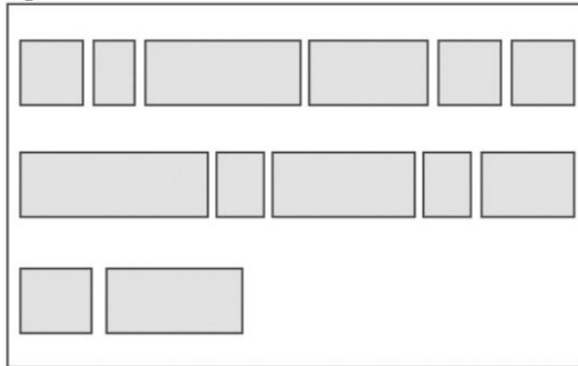
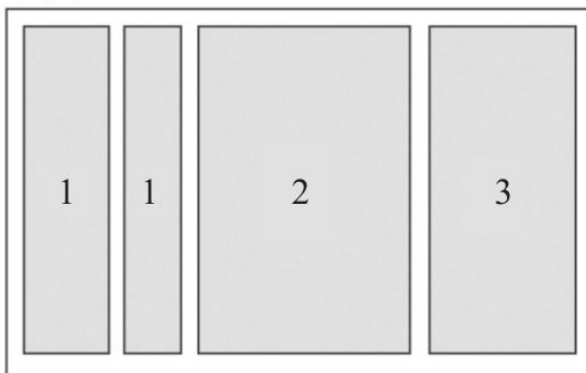
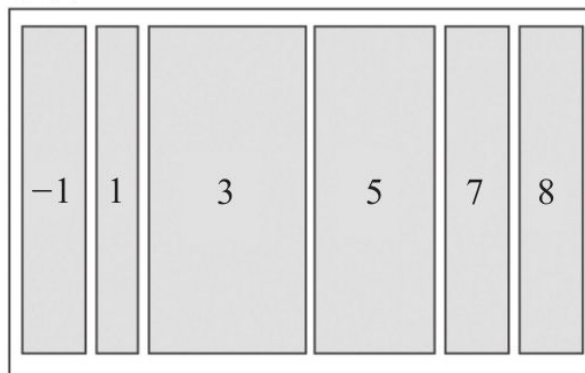


图3-15 align-content示例

示例 1



示例 2



示例 3

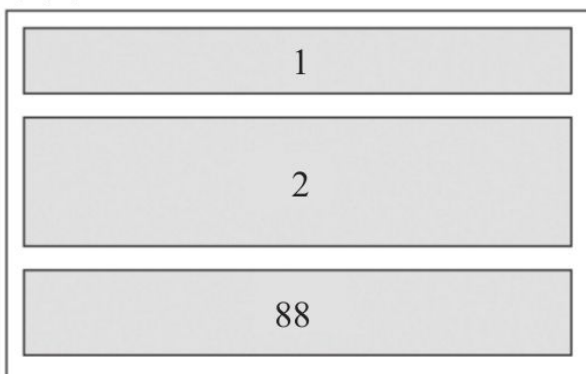


图3-16 order示例

2.flex-grow

flex-grow定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大，语法如下：

```
.myitem{ flex-grow : <number>; }
```

如果所有项目的**flex-grow**值都为1，则它们将等分剩余空间（如果有的话）。如果一个项目的**flex-grow**属性为2，其他项目都为1，则前

者占据的剩余空间将比其他项多一倍，整体按比例填充剩余空间，如图3-17所示。

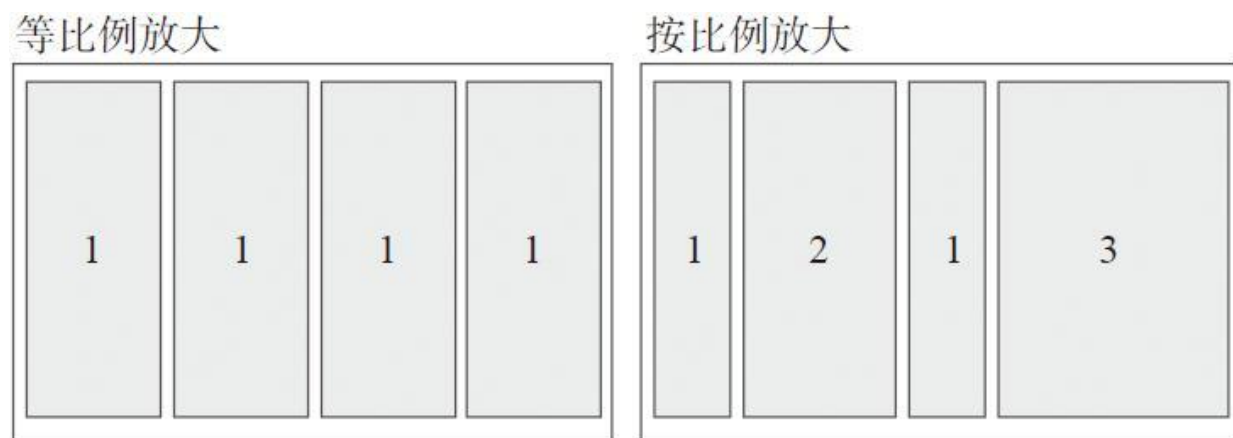


图3-17 flex-grow示例

3.flex-shrink

flex-shrink定义了项目的缩小比例，默认为1，如果空间不足，该项目将缩小，语法如下：

```
.myitem{ flex-shrink : <number>; }
```

如果所有项目的**flex-shrink**属性都为1，当空间不足时，都将等比缩小。如果一个项目的**flex-shrink**属性为0，其他项目都为1，则空间不足时，前者不缩小，负值对该属性无效。这个属性大多数资料都没有细讲，这里我们举个例子（如图3-18），如一个容器宽200px，里面有4个项目，它们的宽度都为60px，那么整体宽度就是 $4 \times 60 = 240\text{px}$ ，比容器多了40px，如果这4个项目的**flex-shrink**值分别为1、2、1、3，那么

它们的宽度分别按比例减少 $40\text{px} \times 1 / (1+2+1+4) = 5\text{px}$ 、 $40\text{px} \times 2 / (1+2+1+4) = 10\text{px}$ 、 $40\text{px} \times 1 / (1+2+1+4) = 5\text{px}$ 、 $40\text{px} \times 4 / (1+2+1+4) = 20\text{px}$ ，缩小后它们的宽度分别为：55px、50px、55px、40px。

4.flex-basis

flex-basis属性用来定义伸缩项目的基准值，剩余的空间将按比例进行缩放。它的默认值为**auto**，即项目的本来大小，语法如下：

```
.myitem{ flex-basis : <length> | auto; }
```

它可以设为跟**width**或**height**属性一样的固定值，如320px，这样项目将占据固定空间。

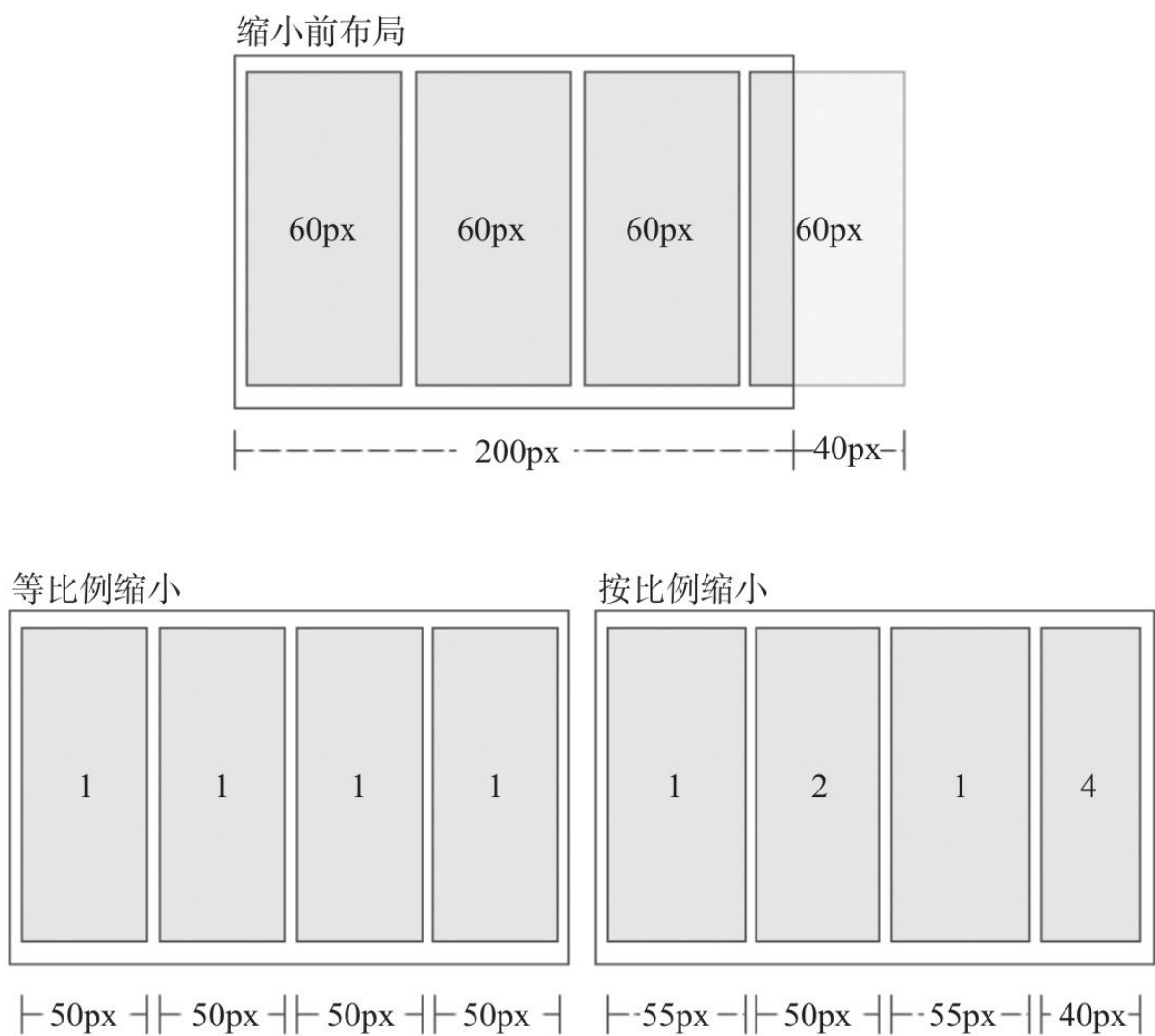


图3-18 flex-shrink示例

5.flex

flex属性是flex-grow、flex-shrink和flex-basis的简写，默认值为0 1 auto。后两个属性可选，语法如下：

```
.myitem{ none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ] }
```

该属性有两个快捷值：**auto**（11 auto）和**none**（00 auto）。

建议优先使用这个属性，而不是单独写三个分离的属性，因为浏览器会推算相关值。

示例代码如下：

```
.myitem{ flex: 1 1 auto }
.myitem{ flex : auto; } /* 同上句 */
.myitem{ flex: 0 0 auto }
.myitem{ flex : none; } /* 同上句 */
.myitem{ flex : 1 auto; }
.myitem{ flex : 1 1; }
```

6.align-self

align-self用来设置单独的伸缩项目在交叉轴上的对齐方式，该属性会复写默认的对齐方式，语法如下：

```
.myitem{ align-self : auto | flex-start | flex-end | center | baseline | stretch;
}
```

该属性6个值除了**auto**，其余和容器**align-items**属性完全一致：

- auto**：表示继承容器**align-items**属性，如果没有父元素，则等同于**stretch**，默认值。

- flex-start**：交叉轴的起点对齐，如图3-19所示。

- flex-end**：交叉轴的终点对齐。

·**center**: 交叉轴的中线对齐。

·**baseline**: 项目根据它们第一行文字的基线对齐。

·**stretch**: 如果项目未设置高度或设置为**auto**，项目将在交叉轴方向拉伸填充整个容器，默认值。

align-self 举例

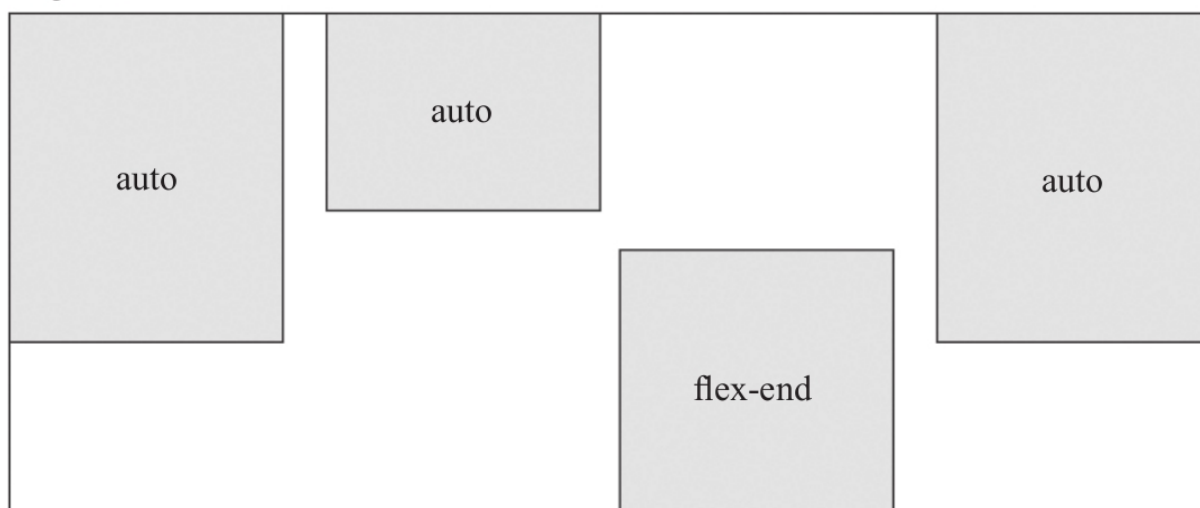


图3-19 align-self示例

3.4 小结

布局是构建页面的基础，如果使用时不小心也会产生一些意料之外的问题。在使用布局时，需要在合适的场景使用合适的定位技术，对元素的定位、重叠、叠放顺序、大小和放置等都要仔细考虑。如果使用浮动，还需要考虑浮动和正常流的关系，在合适的位置还需要清除浮动，这些细节都是在布局中需要考虑的点。下一章我们将讲解组件相关的知识，掌握布局后，我们只需要将组件放置在合适的布局中，即可构建出需要的界面。

第4章 组件

小程序定义了各种各样的组件，它们在WXML中起着各不相同的作用。与HTML元素一样，一个组件是指从组件开始标签到结束标签的所有代码，由于组件可能会被转译为不同端对应的代码，所以在页面创建过程中，不能使用小程序组件标签以外的标签。在本章中，我们将介绍小程序相关组件，包括视图容器、基础容器、表单组件、导航媒体组件、地图、画布、客服会话8大类。为了让大家更好地理解每个组件的特性，避免干扰理解，在本章中我们尽可能少地使用WXSS，所以案例界面不太美观。

4.1 组件定义及属性

上一章介绍了小程序框架原理，在框架基础上官方提供了一系列基础组件，开发者可通过这些基础组件进行任意组合快速开发。小程序组件类似于HTML元素，每个标签代表一个组件，官方对组件作了如下定义：

- 1) 组件是视图层的基本组成单元。
- 2) 组件自带一些功能与微信风格的样式。
- 3) 一个组件通常包括开始标签和结束标签，属性用来修饰这个组件，内容在两个标签之内。

按类型可以将组件划分为七大类：视图容器、基础内容、表单、导航、多媒体、地图、画布。

一个完整的组件结构如下：

```
<tagname property="value">contents</tagname>
```

组件可以通过属性进行配置，属性只能用在开始标签或单个自闭合标签上，不能用于结束标签。一个组件可以对应多个属性，属性具

有名称和值两部分，组件的属性名称都是小写，以连字符“-”连接。组件属性分为所有组件都有的共同属性和组件自定义的特殊属性。

1.组件的共同属性

组件的共同属性指每个组件都有的属性，在每个组件中它们代表的意义和作用都一样，如下所示：

·**id**：组件的唯一表示，保持整个页面唯一。

·**class**：组件里的样式类，在对应的WXSS中定义的样式类。

·**style**：组件的内联样式，可以动态设置的内联样式。使用方式同HTML标签style属性。

·**hidden**：组件是否显示，所有组件默认显示。

·**data-***：自定义属性，组件上触发事件时，会发送给事件处理函数。事件处理函数可以通过**data-scl**获取。

·**bind*/catch***：组件的事件，绑定逻辑层相关事件处理函数。**bind**为冒泡事件，**catch**为非冒泡事件。

除上述属性以外几乎所有组件都有自定义属性，可以对该组件的功能或样式进行修饰，具体参考各个组件的定义。

2.组件的属性类型

每个属性都有其对应的类型，使用时应给属性值传入对应的类型值，属性按类型可分为：

·**Boolean**: 布尔值，组件写上该属性，不管该属性等于什么，其值都为**true**，只有组件上没有写该属性时，属性值才为**false**。如果属性值为变量，变量的值会被转换为**Boolean**类型。

·**Number**: 数字。

·**String**: 字符串。

·**Array**: 数组。

·**Object**: 对象。

·**EventHandler**: 事件处理函数名。

·**Any**: 任意属性。

4.2 视图容器

有过前端经验的同学都了解，在前端项目中我们常使用DIV+CSS进行页面布局，其中<div/>没有任何语义和功能，仅作为容器元素存在，在小程序中，有一套类似<div/>的容器组件，那就是<view/>、<scroll-view/>和<swiper/>。在HTML中大部分标签内部能嵌套任何标签，如<div/>、、<section/>、<p/>等，但是在小程序中，大部分组件都有它自己特殊的功能和意义，标签都有特定的用法，内部也只能嵌套指定的组件，而容器组件内部能嵌套任何标签，容器组件是构建布局的基础组件。本小节主要介绍视图类容器。

4.2.1 view组件

`<view/>`是一个块级容器组件，没有特殊功能，主要用于布局展示，是布局中最基本的UI组件，任何一种复杂的布局都可以通过嵌套`<view/>`组件，设置相关WXSS实现。`<view/>`支持常用的CSS布局属性，如`display`、`float`、`position`甚至Flex布局等，熟悉DIV+CSS布局的读者上手应该很容易。

`<view/>`具备一套关于点击行为的属性：

- `hover`：是否启动点击态，默认值为`false`。

- `hover-class`：指定按下去的样式。当`hover-class="none"`时，没有点击态效果，默认值为`none`。

- `hover-start-time`：按住后多久出现点击态，单位毫秒，默认值为50。

- `hover-stay-time`：手指松开后点击态保留时间，单位毫秒，默认值为400。

为了让大家能更直观的了解`<view/>`布局的特性，下面为大家示范一些常用的布局。

1.水平3栏布局

水平3栏布局如图4-1所示。



图4-1 三栏布局

水平3栏布局思路比较简单，我们在一个<view/>中放置另外3个等分<view/>即能实现，这里我们使用Flex布局实现，代码如下：

```
<view style="display:flex;">
  <view style="background-color:red;flex-grow:1;height:80rpx;"></view>
  <view style="background-color:blue;flex-grow:1;height:80rpx;"></view>
  <view style="background-color:green;flex-grow:1;height:80rpx;"></view>
</view>
```

2.左右混合布局

左右混合布局如图4-2所示。

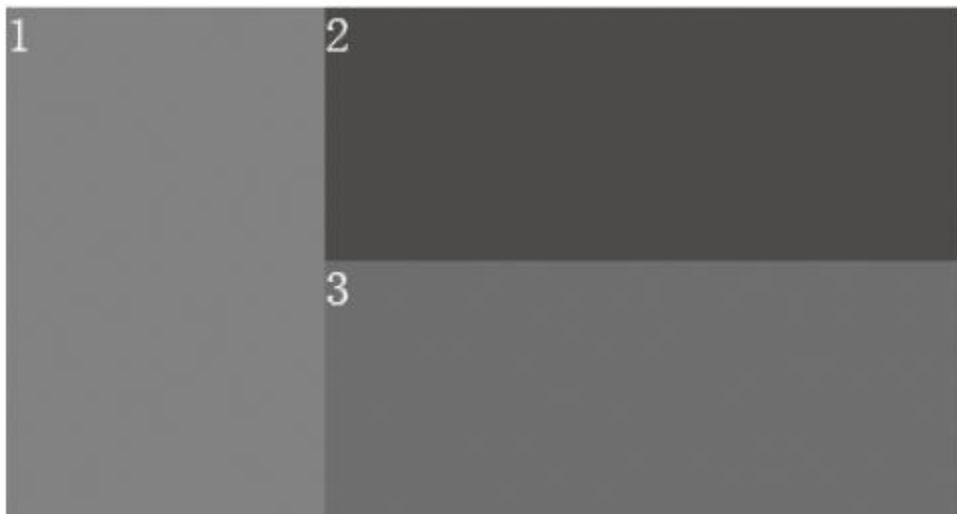


图4-2 左右混合布局

我们先在一个<view/>中水平放置两个<view/>，按需要设置宽度，然后在右侧<view/>中再创建2个<view/>组件，设置为垂直布局，代码如下：

```
<view style="display:flex;height:400rpx;">
  <view style="background-color:red;width:250rpx;color:#fff;">1</view>
  <view style="flex-grow:1;display:flex;flex-direction:column;">
    <view style="flex-grow:1;background-color:blue;color:#fff;">2</view>
    <view style="flex-grow:1;background-color:green;color:#fff;">3</view>
  </view>
</view>
```

3.上下混合布局

上下混合布局如图4-3所示。



图4-3 上下混合布局

同左右混合布局思路一样，我们先在一个<view/>中放置两个垂直布局<view/>，按需要设置高度，然后在下面的<view/>中再创建两个<view/>，设置为水平布局，代码如下：

```
<view style="display:flex;flex-direction:column;height:400rpx;">
  <view style="background-color:red;height:150rpx;color:#fff;">1</view>
  <view style="flex-grow:1;display:flex;">
    <view style="flex-grow:1;background-color:blue;color:#fff;">2</view>
    <view style="flex-grow:1;background-color:green;color:#fff;">3</view>
  </view>
</view>
```

通过上面3个案例大家可以发现，任何复杂的布局都是通过不断嵌套<view/>实现的，在小程序中使用<view/>就像我们在HTML使用<div/>一样便捷，通过<view/>我们可以构建出任何想要的界面。

4.2.2 scroll-view组件

在布局过程中，我们需要一些容器具备可滑动的能力，尽管我们可以通过给<view/>设置overflow: scroll属性来实现，但由于小程序实现原理中没有DOM概念，我们没法直接监听<view/>滚动、触顶、触底等事件，这时便需要使用<scroll-view/>。<scroll-view/>在<view/>基础上增加了滚动相关属性，通过设置这些属性，我们能响应滚动相关事件。<scroll-view/>属性如下：

- scroll-x: 允许横向滚动，默认为false。
- scroll-y: 允许纵向滚动，默认为false。
- upper-threshold: 距顶部/左边多远时（单位px），触发scrolltoupper事件，默认值为50。
- lower-threshold: 距底部/右边多远时（单位px），触发scrolltolower事件，默认值为50。
- scroll-top: 设置竖向滚动条位置。
- scroll-left: 设置横向滚动条位置。

·**scroll-into-view**: 值应为某子元素id, 滚动到该元素时, 元素顶部对齐滚动区域顶部。

·**bindscrolltoupper**: 滚动到顶部/左边, 会触发scrolltoupper事件。

·**bindscrolltolower**: 滚动到底部/右边, 会触发scrolltolower事件。

·**bindscroll**: 滚动时触发, `event.detail={scrollLeft, scrollTop, scrollHeight, scrollWidth, deltaX, deltaY}`。

目前, 有些组件不能在`<scroll-view/>`中使用: `<textarea/>`、`<video/>`、`<map/>`、`<comvas/>`。

下面我们通过`<scroll-view/>`创建一个纵向滚动区域, 并监听它的滚动事件。需要注意的是, 在使用纵向滚动时, 需要先给`<scroll-view/>`一个固定高度, 如图4-4所示。

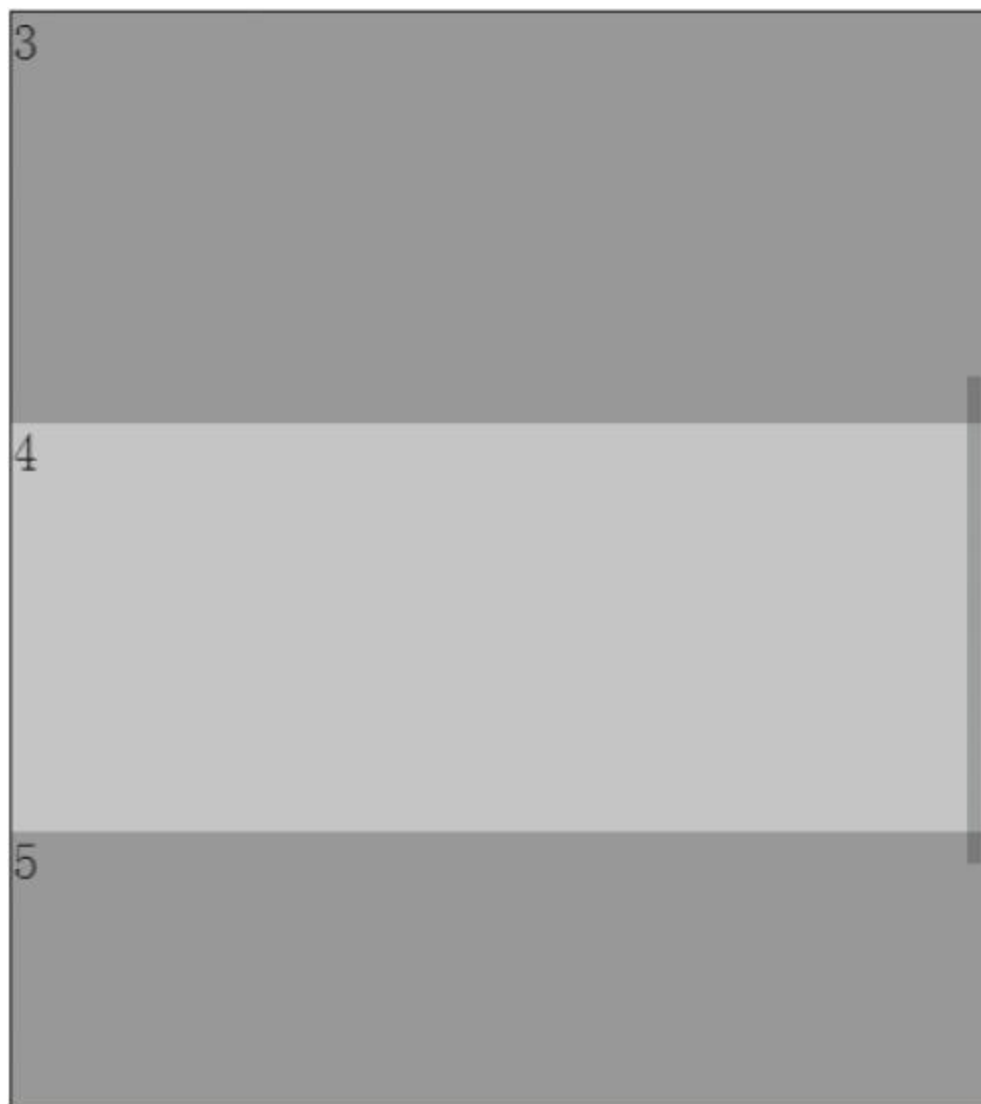


图4-4 可滚动视图区

代码如下:

```
<scroll-view class="scroll-container" upper-threshold="0"
  lower-threshold="100" scroll-into-view="{{toView}}"
  bindscroll="scroll" bindscrolltolower="scrollToLower"
  bindscrolltoupper="scrollToUpper" scroll-y="true"
  scroll-top="{{scrollTop}}">
  <view id="item-1" class="scroll-item bg-red">1</view>
  <view id="item-2" class="scroll-item bg-blue">2</view>
  <view id="item-3" class="scroll-item bg-red">3</view>
  <view id="item-4" class="scroll-item bg-blue">4</view>
  <view id="item-5" class="scroll-item bg-red">5</view>
  <view id="item-6" class="scroll-item bg-blue">6</view>
</scroll-view>
```

```

<view class="act">
<button bindtap="scrollToTop">点击滚动到顶部</button>
</view>

/*给予固定高度*/
.scroll-container {
  border : solid 1px;
  height : 800rpx;
}

.scroll-container .scroll-item {
  height : 300rpx;
  width : 120%;
}

.bg-blue { background-color: #87CEFA; }
.bg-red { background-color: #FF6347; }

.act { padding : 10px; }

Page( {
  data : {
    toView : 'item-3'//第一次渲染时, <scroll_view>默认滚动到id值为"item-3"区域
  },

  scrollToUpper : function() {
    console.log( '触发到滚动顶部事件' );
  },

  scrollToLower : function() {
    console.log( '触发滚动到底部事件' );
  },
  //点击按钮时, 滚动到顶部
  scroll : function() {
    console.log( '触发了滚动事件' );
  },

  scrollToTop : function() {
    this.setData( {
      scrollTop : '0'
    } );
  }
} );

```

布局页面时，如对事件没有特殊要求，也可以使用<view/>代替<scroll-view/>进行布局。

4.2.3 滑块视图组件

滑块视图容器在前端开发中是一个常见组件，利用它我们可以实现轮播图、滑动页面、图片预览等效果。一个完整的滑块视图组件由`<swiper/>`和`<swiper-item/>`两个标签组成，它们不能单独使用，一个`<swiper/>`中只能放置一个或多个`<swiper-item/>`，放置其他节点会被删除，`<swiper-item/>`内部能放置任何组件，默认宽高自动设置为100%。`<swiper-item/>`组件作为容器没有任何特殊属性，`<swiper/>`组件属性如下：

- `indicator-dots`：是否显示面板指示点，默认为`false`。
- `autoplay`：是否自动切换，默认为`false`。
- `current`：当前所在页面的`index`，默认为0。
- `interval`：自动切换时间间隔，默认为5000。
- `duration`：滑动动画时长，默认为1000。
- `circular`：是否采用衔接滑动，默认值为`false`。
- `bindchange`：`current`改变时会触发`change`事件，`event.detail={current: current}`。

1.基本轮播图

本案例主要讲解滑块视图最基本用法，开始图如图4-5所示。



图4-5 简单轮播示例

示例代码如下：

```
<1- 开启默认面板指示点，并设置为自动播放-->  
<swiper class="banner" indicator-dots="true" autoplay="{{autoplay}}"
```

```

        current="0" interval="2000" duration="300" bindchange="change">
        <block wx:for="{{sliderList}}">
            <swiper-item class="{{item.className}}">{{item.name}}</swiper-item>
        </block>
    </swiper>

    <view>
        <button bindtap="play">暂停|播放</button>
    </view>

    .banner { height : 400px; background-color: #ddd; }

    .bg-blue { background-color: #87CEFA; }
    .bg-red { background-color: #FF6347; }
    .bg-green { background-color: #43CD80; }

    Page( {
        data : {
            autoplay : true,
            sliderList : [
                { className : 'bg-red', name : 'slider1' },
                { className : 'bg-blue', name : 'slider2' },
                { className : 'bg-green', name : 'slider3' }
            ]
        },
        play : function() {
            this.setData( {
                autoplay : !this.data.autoplay
            } );
        },
        change : function() {
            console.log( '执行了滚动' );
        }
    } );

```

2.自定义轮播图

`<swiper>`默认组件中不提供面板指示点的样式设置，而在轮播组件中，我们常常需要设置自定义面板指示点，这时可以通过`bindchange`事件实现图4-6一样的定义面板。



图4-6 自定义轮播

代码如下:

```
<view class="customSwiper">
  <!--不开启默认面板-->
  <swiper class="banner" autoplay="true" interval="2000" duration="300"
    bindchange="switchTab">
    <block wx:for="{{sliderList}}">
      <swiper-item>
        <image style="width:100%;height:100%;" src="{{item.imageSource}}"/>
      </swiper-item>
    </block>
  </swiper>
  <!--自定义面板构建-->
  <view class="tab">
    <block wx:for="{{sliderList}}">
      <view wx:if="{{item.selected}}" class="tab-item
        selected">{{index+1}}</view>
      <view wx:else class="tab-item">{{index+1}}</view>
    </block>
  </view>
</view>
.customSwiper { height : 379.5rpx; position: relative; }
.customSwiper swiper { height : 100%; }
/*自定义面板样式*/
.tab { height : 70rpx; position: absolute; bottom : 0; display: flex;
  width : 100%; text-align: center; justify-content : center;
  align-items: center; }
.tab .tab-item { background-color: #ccc; height : 50rpx; width : 50rpx;
  line-height: 50rpx; font-size : 12rpx; color : #fff; border-radius: 4px;
  margin-right : 10px; }
.tab .tab-item.selected { background-color: red; }
Page( {
  data : {
```



```
        sliderList : [
            { selected : true, imageSource : './image/banner1.jpg' },
            { selected : false, imageSource : './image/banner2.jpg' },
            { selected : false, imageSource : './image/banner3.jpg' }
        ]
    },
    //监听swiper滚动事件，并切换面板
    switchTab : function( e ) {
        var sliederList = this.data.sliderList,
            i, 1;
        //修改指示点选中态
        for ( i = 0; item = sliederList[i]; ++i ) {
            item.selected = e.detail.current == i;
        }
        this.setData( {
            sliderList : sliederList
        } );
    }
} );
```

4.3 基础组件

4.3.1 icon

`<icon/>`是页面中非常常用的组件，它通常用于表示状态，起到引导作用。在HTML中，我们通常通过修改元素样式、添加图片等方案实现图标，在`<icon/>`中，官方为大家提供了一套符合微信设计规范的样式类型，当然我们也可以通过自定义样式创建自定义图标。`<icon/>`有以下属性：

- `type`: icon的类型。有效值包括：success、success_no_circle、info、warn、waiting、cancel、download、search、clear。

- `size`: icon的大小，单位px。默认值为23px。

- `color`: icon的颜色，同CSS的color。

1. 示例

`<icon/>`的使用十分简单，下面为大小不同、不同类型、不同颜色的图标展示，如图4-7所示。

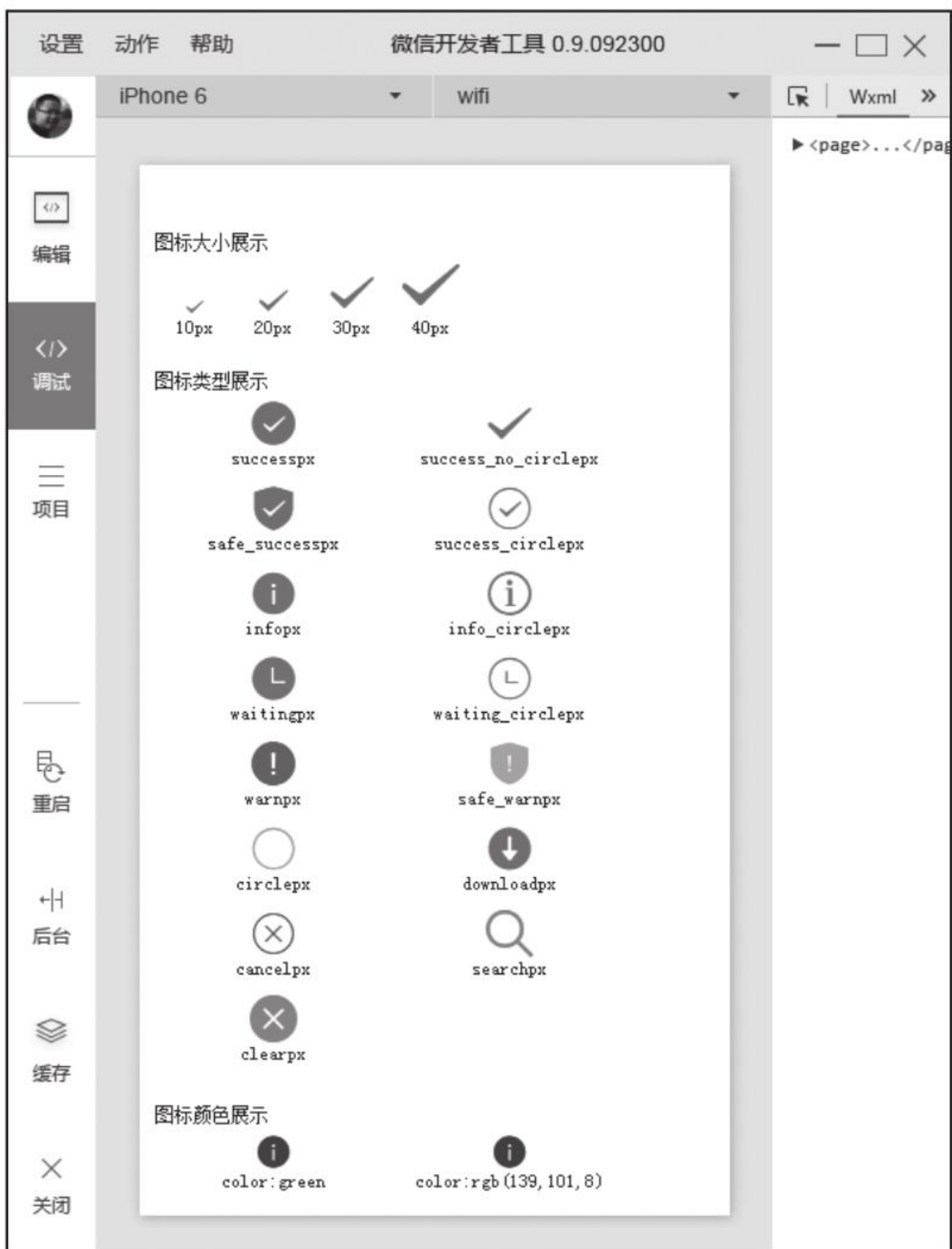


图4-7 icon示例

代码如下:

```
<view class="section size">
  <view class="title">图标大小展示</view>
  <view class="list">
    <block wx:for="{{sizeList}}">
      <view class="item">
        <icon type="success_no_circle" size="{{item}}"/>
        <label>{{item}}px</label>
      </view>
    </block>
  </view>
</view>
<view class="section">
  <view class="title">图标类型展示</view>
  <view class="list">
    <block wx:for="{{typeList}}">
      <view class="item">
        <icon type="{{item}}" size="30"/>
        <label>{{item}}px</label>
      </view>
    </block>
  </view>
</view>
<view class="section">
  <view class="title">图标颜色展示</view>
  <view class="list">
    <block wx:for="{{colorList}}">
      <view class="item">
        <icon type="info" color="{{item}}"/>
        <label>color:{{item}}</label>
      </view>
    </block>
  </view>
</view>
</view>

.section{ font-size : 12px; padding : 10px; }
.section .list{ display:flex; flex-wrap : wrap; }
.section .list .item { width : 300rpx; padding : 5px 0; display: flex;
                        flex-direction:column; justify-content:flex-end; }
.section.size .list .item { width : 100rpx; padding : 0; }
.section .list .item label { text-align: center; }
.section .list .item icon { text-align: center; }

Page( {
  data : {
    /*不同大小*/
    sizeList : [10, 20, 30, 40],
    /*不同类型*/
    typeList : ['success','success_no_circle', 'safe_success',
                'success_circle', 'info','info_circle', 'waiting',
                'waiting_circle', 'warn', 'safe_warn', 'circle', 'download',
                'cancel', 'search', 'clear'],
    /*不同颜色*/
    colorList : ['green', 'rgb(139,101,8)']
  },
} );
```

2.自定义样式<icon/>组件

虽然官方为大家预设了一些常用的样式，但是在项目中我们常常会使用自定义的UI风格，这时就需要创造自定义样式的<icon/>组件，图4-8所示的案例中我们将通过固定<icon/>大小、位移、背景图实现自定义图标的效果，如图4-9所示。这种方式在网页布局中十分常见，在小程序中的使用也完全一致。虽然这种实现方式适合大部分标签，但使用<icon/>标签更符合语义。

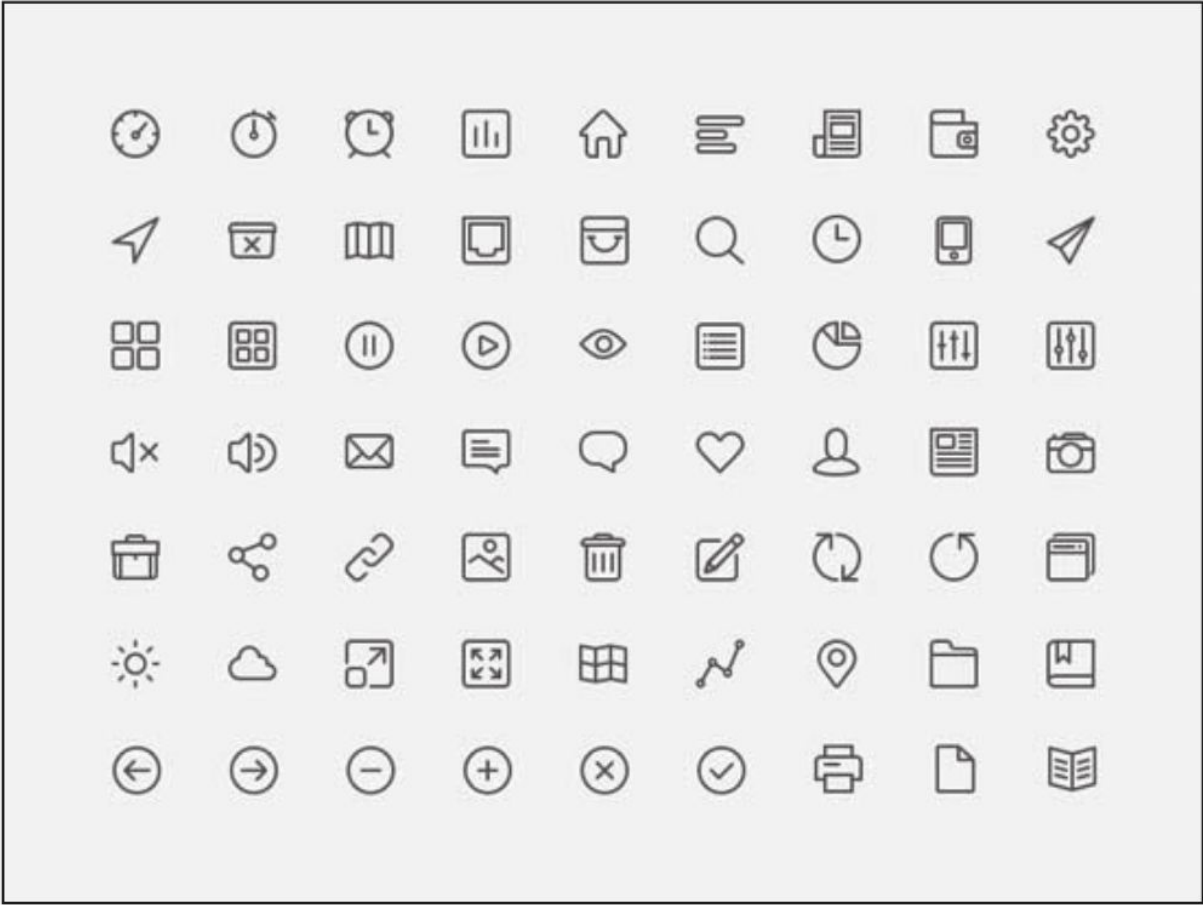


图4-8 图标背景图

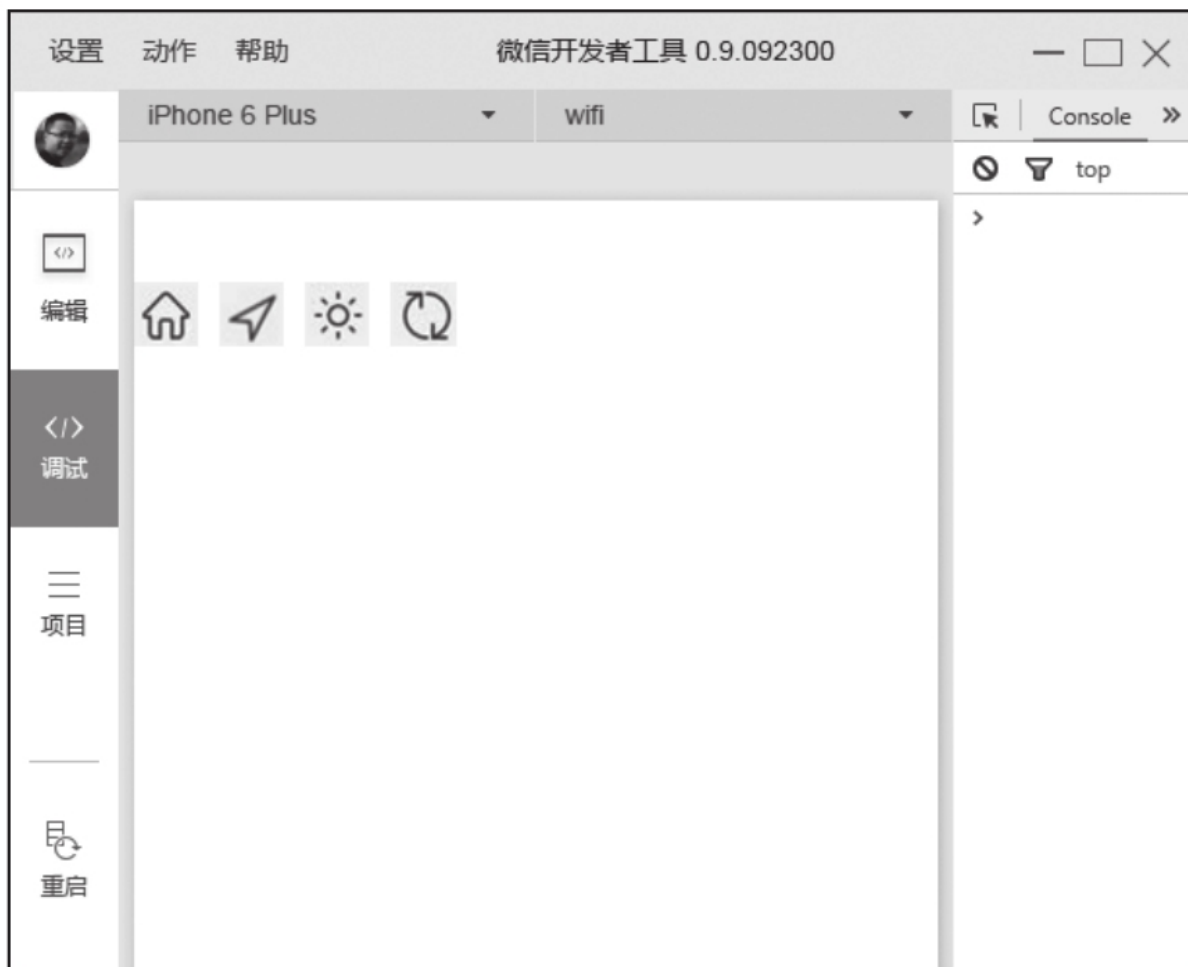


图4-9 自定义icon

代码如下:

```
<icon class="myicon home"/>
<icon class="myicon arrow"/>
<icon class="myicon sun"/>
<icon class="myicon refresh"/>

/*固定背景图与图标大小*/
.myicon{ background: url( ./image/icons.jpg ) no-repeat; background-size:
    1100rpx 826rpx; width : 60rpx; height : 60rpx; }
/*移动背景位置, 实现不同icon*/
.myicon.arrow{ background-position: -90rpx -185rpx; }
.myicon.home{ background-position: -520rpx -90rpx; }
.myicon.sun{ background-position: -90rpx -575rpx; }
.myicon.refresh{ background-position: -730rpx -476rpx; }

.myicon { margin-right: 20rpx; }
```

示例中，根据微信提供的特性我们使用了rpx单位，让<icon/>具有自适用功能。<icon/>是项目中常用的组件，开发过程中应尽量使用小程序提供的系统样式，这能保证小程序在微信系统中UI风格的一致性。对于某些特殊情况，我们可以在系统图标基础上进行拓展，满足开发需要。

4.3.2 text组件

`<text/>`组件主要用于文本内容的展示，类似HTML中`<p/>`标签，在小程序中，只有`<text/>`节点内部的内容能被长按选中，大家在展示文本的时候一定要根据这个特性注意场景的使用。`<text/>`是行内元素，除了组件共同属性外并没有提供其他属性，文本中的内容支持转义字符“\”，常用的转义字符可以参考网络资料。`<text/>`组件内只支持`<text/>`嵌套，这样我们能通过嵌套`<text/>`实现某些字符加粗、标红等特殊样式的设置。

`<text/>`用法非常简单，将文本放置到`<text>`标签中间即可，在示例中“\n”、“\t”转义符都都被正常转译，如图4-10所示。

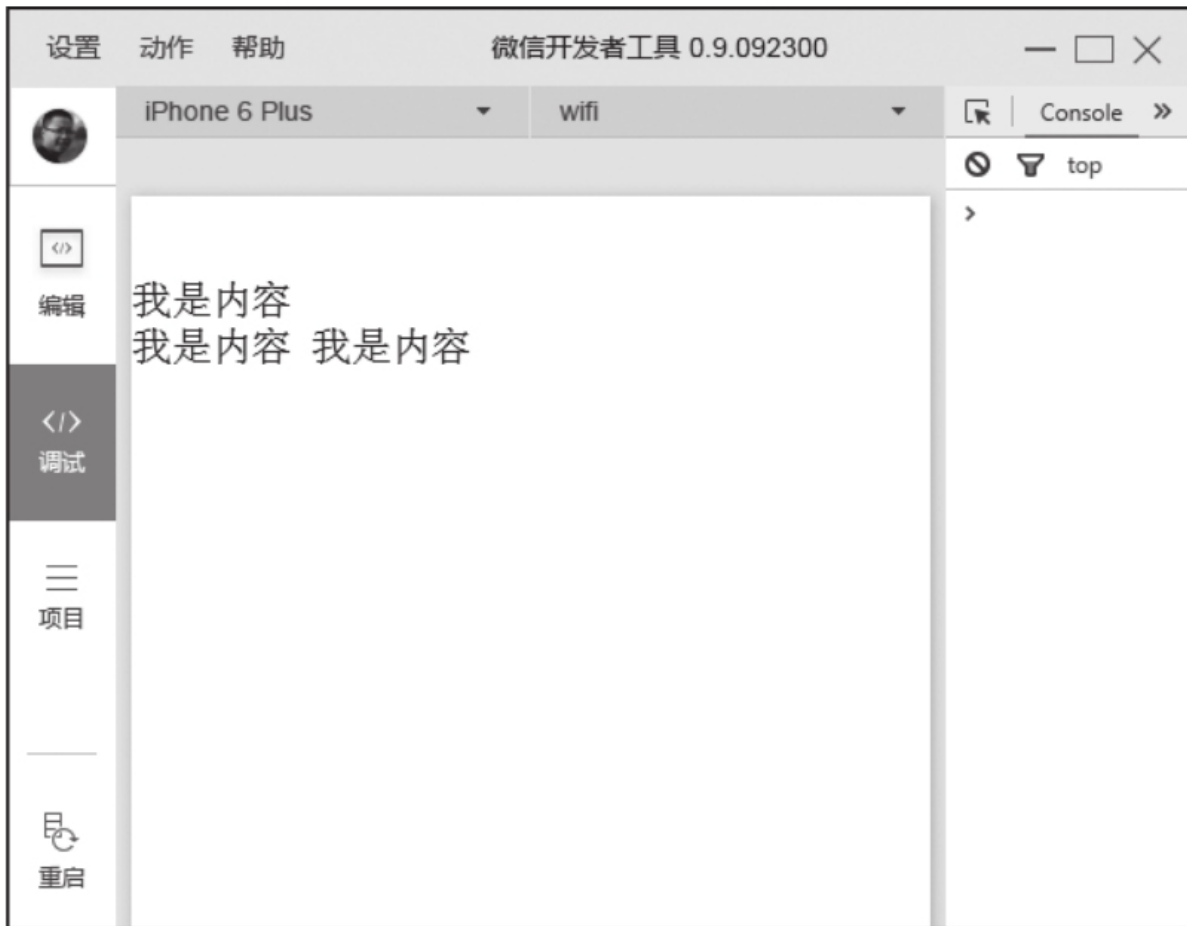


图4-10 text组件示例

代码如下：

```
<text>{{content}}</text>

Page( {
  data : {
    content : '我是内容\n我是内容\t我是内容'
  }
} );
```

小程序中没有等标签，不过我们可以通过<text/>嵌套、设置样式达到同样的效果。

4.3.3 progress组件

`<progress/>`用于显示进度状态，比如资源加载、用户资料完成度、媒体资源播放进度等。目前，`<progress/>`是通用的进度条组件，我们也可以利用各种布局、样式实现一套自定义进度条。`progress`是块级元素，属性如下：

- `percent`: 当前进度占有所有进度的百分比，取值区间为0到100。
- `show-info`: 是否在进度条右侧显示百分比，默认为`false`。
- `stroke-width`: 进度条线的宽度，单位`px`，默认值为6。
- `color`: 进度条颜色，默认值为`#09BB07`。
- `active`: 渲染时是否开启进度条从左到右的动画，默认值为`false`。开启后每次修改`percent`触发进度条重新渲染，都会从左到右显示动画。

下面简单展示进度条的相关属性，如图4-11所示。

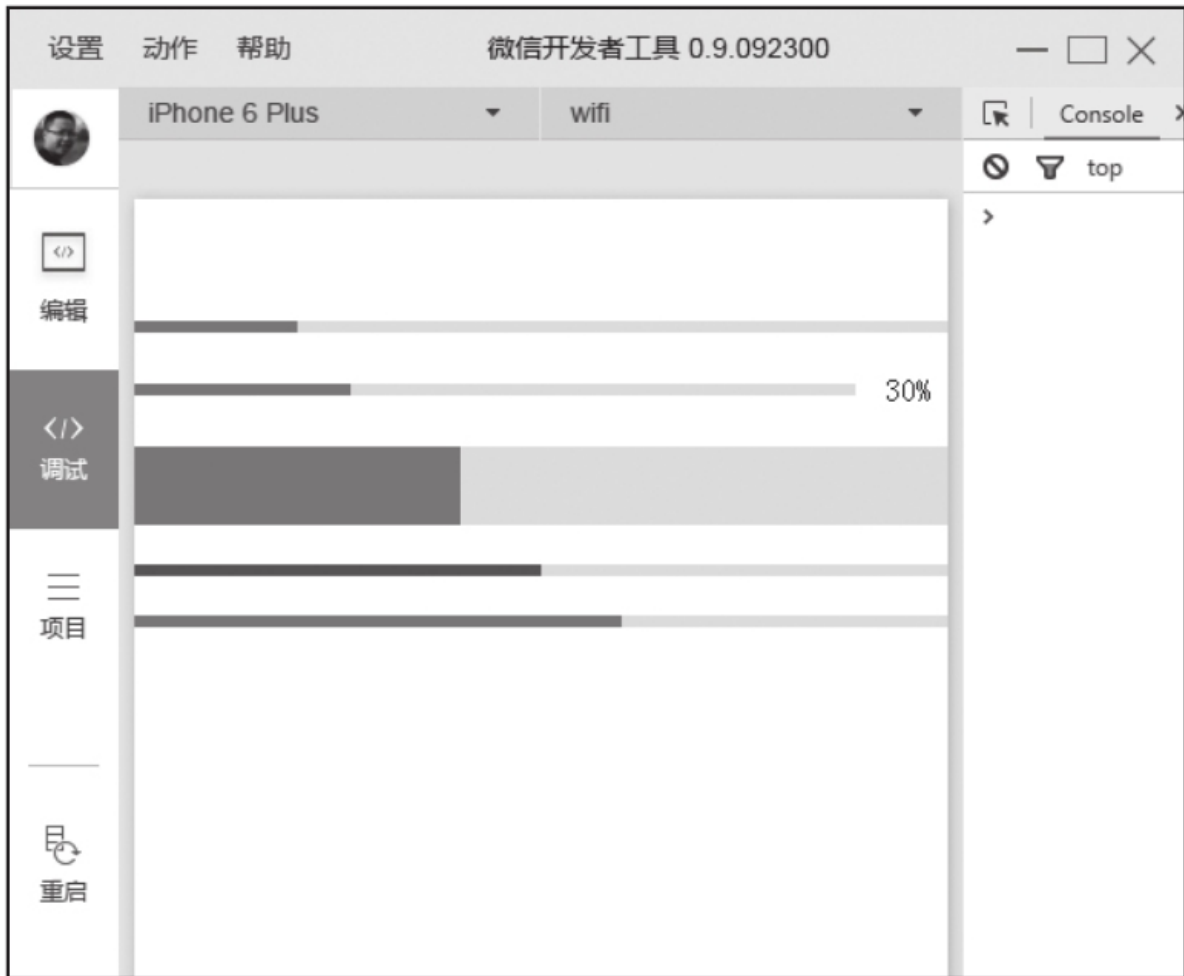


图4-11 progress示例

代码如下:

```
<progress percent="20"></progress>
<progress percent="30" show-info="true"></progress>
<progress percent="40" stroke-width="40"></progress>
<progress percent="50" color="#CD5555"></progress>
<progress percent="60" active></progress>
```

4.4 表单组件

表单是应用中获取用户输入的重要手段，它对于系统极其重要，用户在应用中输入的大部分内容都是在表单元素中完成的。本节主要为大家介绍表单相关组件、特性、组件之间的关系。如何将数据上传至后台会在第5章中介绍。本节中提到的表单组件不仅可以放置在<form/>标签中使用，同时也可以作为单独组件和其他组件混合使用。

4.4.1 radio组件

单项框可以用来生成一组单选按钮，供用户从一批固定的选项中选择，它适合于可用有效数据不多的情况，小程序中单选框是由`<radio-group>`（单项选择器）和`<radio>`（单选项目）两个组件组合而成，一个包含多个`<radio>`的`<radio-group>`表示一组单选项，在同一组单选项中的`<radio>`是互斥的，当一个按钮被选中，之前选中的按钮就变为非选中。当需要用户在待选项中选择唯一的答案时，就需要使用到单项框。

1.radio-group

在小程序中`<radio>`不能单独使用，同一组`<radio>`需要包含在一个`<radio-group>`中，这样才能形成一组单项选择按钮，`<radio>`的选中态不能直接获取，需要通过`<radio-group>`的`change`事件进行获取。`<radio-group>`内部除了包含`<radio>`也可以包含其他标签，它更像是一个看不见的容器，当包含其他标签时，也仅仅对标签内部的`<radio>`产生影响，而不影响其他组件。`<radio-group>`仅有一个属性：

bindchange: 绑定`<radio-group>`change事件，`<radio-group>`中的选中项发生变化时触发change事件，`event.detail={value: 选中项radio的value}`。

2.radio

`<radio/>`是`<radio-group/>`中的一个单选按钮，具有以下属性：

- `value`: `<radio/>`标识。当该`<radio/>`选中时，`<radio-group/>`的`change`事件会携带`<radio/>`的`value`。
- `checked`: 当前`<radio/>`是否选中，一个`<radio-group/>`中只能有一个`<radio/>`的`checked`为`true`，如果设置多个，将默认选中最后一个为`true`的单选项，默认为`false`。
- `disabled`: 是否禁用，禁用后不能点击，默认为`false`。
- `color`: `radio`的颜色，同CSS的`color`。

3.示例

这里为大家编写一组单选项，以展示`<radio-group/>`和`<radio/>`的关系，如图4-12所示。

代码如下：

```
<radio-group bindchange="changeChooosed">
  <view wx:for="{{radios}}">
    <radio value="{{item.value}}" checked="{{item.checked}}"/>{{item.text}}
  </view>
</radio-group>
Page( {
  data : {
    radios : [
      {value : '1',text : '选项1',checked : false },
      {value : '2',text : '选项2', checked : true},
      {value : '3',text : '选项3', checked : false},
      {value : '4',text : '选项4', checked : false}
```

```
    ]  
  },  
  changeChoosed : function( event ) {  
    console.log( '你选中了: ' + event.detail.value );  
  }  
} );
```

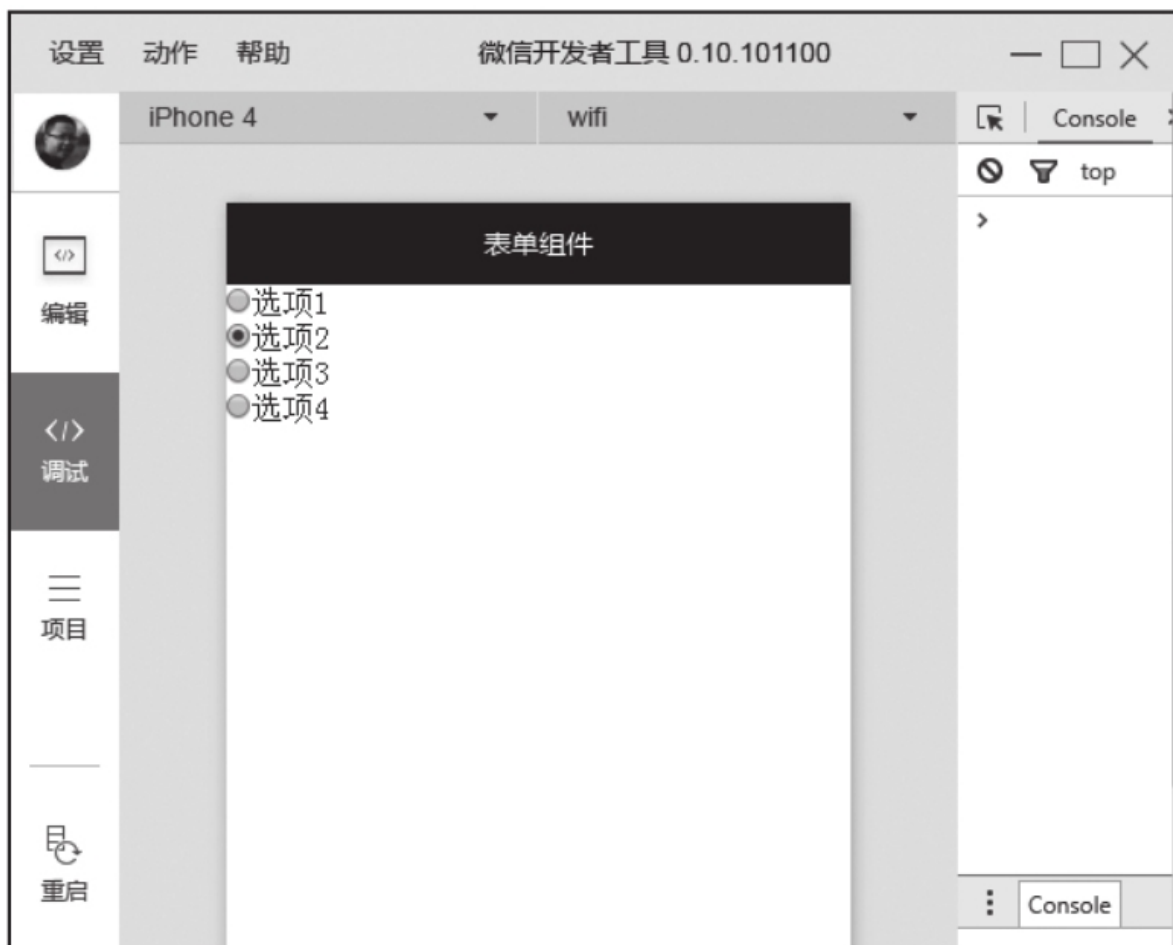


图4-12 radio示例

点击案例中的单选框会发现，调试工具控制台会打印出当前选中的项的value值。在这个案例中点击文案是不能选中单选框的，这种体验略差，我们可以使用<label/>对其进行优化，具体使用方式可以参考label组件。

4.4.2 checkbox组件

与单选框一样，小程序中的复选也是由<checkbox-group/>（多项选择器）和<checkbox/>（多选项目）两个组件组合而成。一个包含多个<checkbox/>的<checkbox-group/>表示一组多选项，一组多选项允许在待选项中选中一项以上的选项。

1.checkbox-group

与<radio-group/>一样，<checkbox-group/>用于包裹<checkbox/>，而且仅有一个属性bindchange：绑定<checkbox-group/>change事件，<checkbox-group/>中的选中项发生变化时触发change事件，event.detail={value: [选中项checkbox的value数组]}。

2.checkbox

<checkbox/>是<checkbox-group/>中的一个多选项目，它的属性和<radio/>一样：

- value: <checkbox/>标识，选中时触发<checkbox-group/>的change事件，并携带<checkbox/>的value。

- checked: 当前<checkbox/>是否选中，可用来设置默认值，一个<checkbox-group/>允许一个或多个<checkbox/>的checked为true，默认

为false。

·disabled: 是否禁用，禁用后不能点击，默认为false。

3.示例

我们通过示例为大家演示复选框的使用，如图4-13所示。

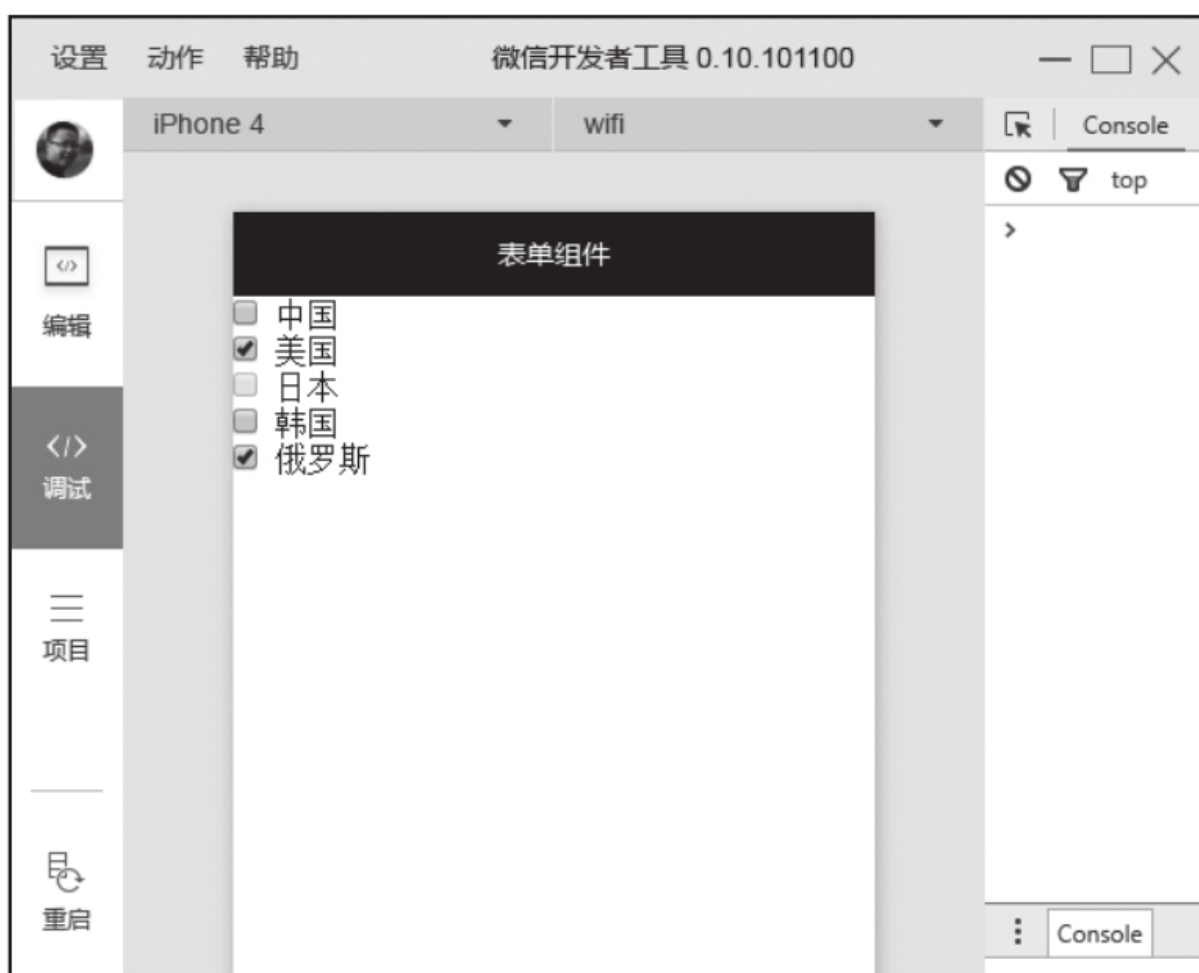


图4-13 复选框示例

代码如下：

```
<checkbox-group bindchange="checkboxChange">
  <view wx:for="{{countryys}}">
    <checkbox value="{{item.value}}" checked="{{item.checked}}"
      disabled="{{item.disabled}}" />
      {{item.name}}
    </view>
  </checkbox-group>

Page( {
  data : {
    countryys : [
      { name : '中国', value : '1' },
      { name : '美国', value : '2', checked : true },
      { name : '日本', value : '3', disabled : true },
      { name : '韩国', value : '4' },
      { name : '俄罗斯', value : '5', checked : true }
    ]
  },

  checkboxChange : function( e ) {
    console.log( '你选中的项目有: ' + e.detail.value );
  }
} );
```

整体来看复选框和单选框的使用方式基本一致，最大的区别在于交互上单选框在同一组单选项中只能选中一个，而复选框可以选中多个；选中状态切换时`change`方法传入的参数值一个是传入单值，一个是传入数组。大家可以将两个组件对比学习。

4.4.3 switch组件

`<switch/>`是一个可以在两种状态切换的开关选择器，现在很多APP都在使用，最常见的就是iOS或Android的系统开关。`<switch/>`在功能上和`<checkbox/>`有点接近，不同点在于`<switch/>`是个单独控件，而`<checkbox/>`是由多个单选项组合而成，在小程序中当`<switch/>`的`type`属性值为`checkbox`时，它的UI表现和`<checkbox/>`非常接近。`switch`属性如下：

- `checked`：是否选中，默认为`false`。
- `type`：`<switch/>`的UI样式，有效值为`switch`、`checkbox`，默认为`switch`。
- `bindchange`：`checked`改变时触发`change`事件，`event.detail={value: checked}`。

`<switch/>`非常容易理解，大家主要了解它的两种UI状态，示例如图4-14所示。

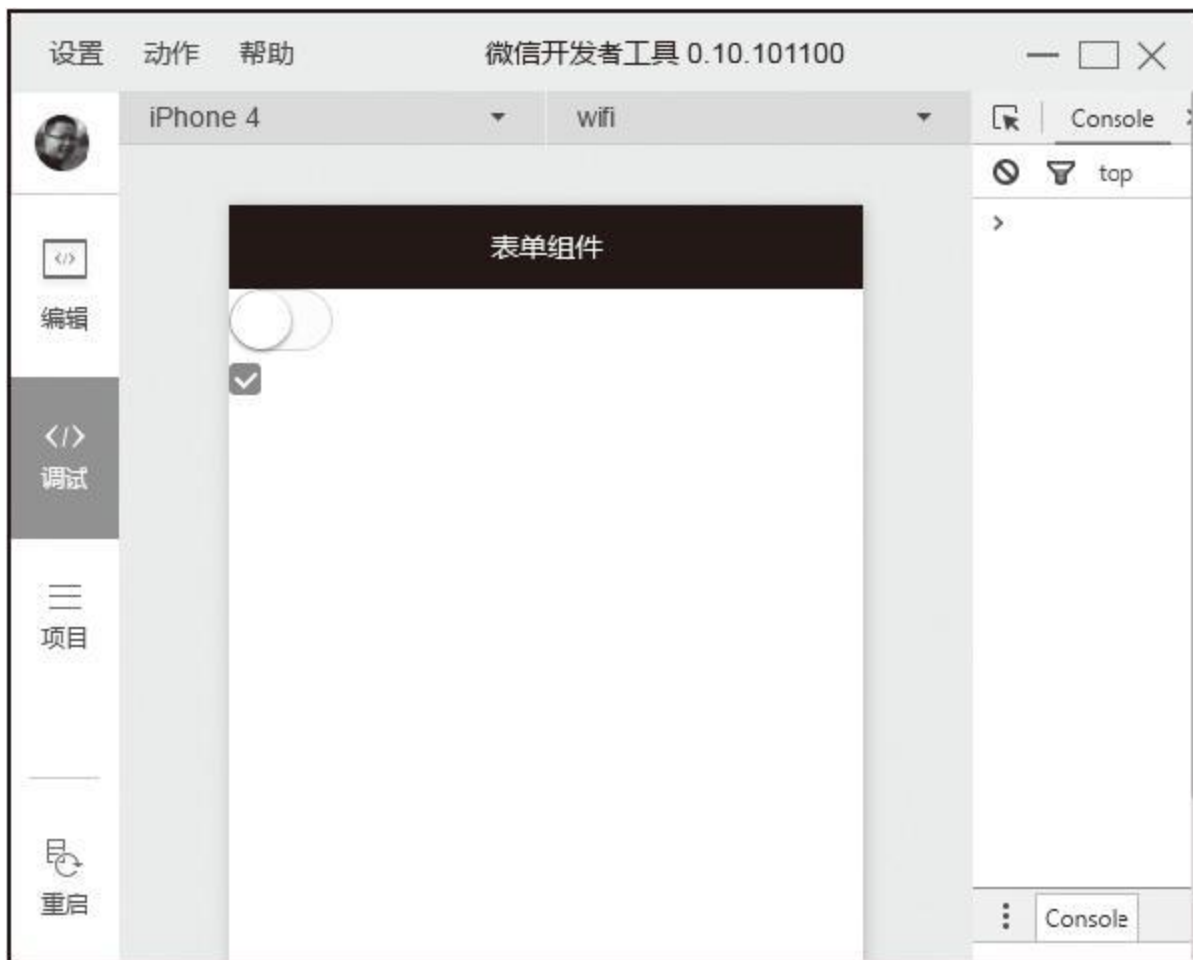


图4-14 switch示例

代码如下：

```
<view wx:for="{{list}}">
  <switch data-name="{{item.name}}" type="{{item.type}}"
    checked="{{item.checked}}" bindchange="{{item.changeEventName}}"/>
</view>

Page( {
  data : {
    switchs : [
      {
        name : 'switch1',
        checked : false,
        type : 'switch', /* 滑块样式 */
        changeEventName : 'change'
      },
      {
        name : 'switch2',
        checked : true,
```

```
        type : 'checkbox', /* 选择框样式 */
        changeEventName : 'change'
    }
    ],
},
// 修改开关选择器对象模型的选中值
change : function( e ) {
    var name = e.currentTarget.dataset.name,
        switchs = this.data.switchs,
        i, s;

    for ( i = 0; s = switchs[i]; ++i ) {
        if ( s.name == name ) {
            s.checked = e.detail.value;
            break;
        }
    }
    console.log( name + '的选中态为: ' + e.detail.value );
}
} );
```

示例change方法中我们根据用户的操作反向修改了数据模型中对应的值，页面进行数据绑定后保证数据状态和页面状态一致是非常关键的，在项目过程中大家一定要注意。

4.4.4 label组件

在<radio/>和<checkbox/>案例中，点击文案时不能选中对应的单选框或复选框，这时我们可以利用<label/>改进表单组件的可用性，通过绑定for属性让用户点击<label/>时触发对应的控件，目前可以绑定的控件有<button/>、<checkbox/>、<radio/>、<switch/>。

小程序中<label/>的触发规则有两种：

- 将控件放在标签内。当用户点击时触发中第一个控件。
 - 设置<label/>的for属性。当用户点击时触发for属性对应的控件。
- for属性优先级高于内部控件。

<label/>只有一个for属性：绑定控件的id。

下面的示例（如图4-15所示）只是为了展示label的特性，一些奇怪的使用方式并不推荐在项目中使用。



图4-15 label示例

代码如下:

```
<view class="section">
  <!-- 点击文案 仍然能触发<checkbox-group/>的change事件 -->
  <checkbox-group bindchange="checkboxchange">
    <label>
      <checkbox value="value-1"/> 点击文案也能选中radio
    </label>
  </checkbox-group>
</view>

<view class="section">
  <label>
    <switch/> 点击文案切换开关
  </label>
</view>

<view class="section">
  <view class="title">选中元素内第一个</view>
```

```

<checkbox-group>
  <label>
    <checkbox value="value-1"/>第一个
    <checkbox value="value-2"/>第二个
    <checkbox value="value-3"/>第三个
    <checkbox value="value-4"/>第四个
    <checkbox value="value-5"/>第五个
  </label>
</checkbox-group>
</view>

<view class="section">
  <view class="title">for属性优先级大于内部元素</view>
  <checkbox-group>
    <label for="mycheckbox">
      <checkbox/>label内部元素
      <checkbox/>label内部元素
    </label>
    <checkbox id="mycheckbox"/>label外部元素
  </checkbox-group>
</view>

.section { font-size : 12px; padding: 20px 10px; border-top : solid 1px #eee; }
.section .title { padding : 5px 0; margin-bottom: 5px; display : block; }
.section input { border : solid 1px #ccc; background-color: #fff; border-radius: 4px; }

Page( {
  checkboxchange : function( event ) {
    console.log( event.detail.value );
  }
} );

```

上例中，<label/>元素内嵌套了多个组件，我们点击<label/>中所有字符都选中第一个控件，但是点击单个<checkbox/>仍然可以修改其选择状态，这样的交互体验是混乱的，所以在实际项目中，一个<label/>尽量只包裹一个组件。

4.4.5 slider组件

滑动选择器是一种在移动端常用的交互组件，大家还记得手机上的亮度调节工具吗？那就是滑动选择器，在小程序中我们可以利用 `<slider/>` 快速生成一个符合系统UI的滑动选择器。滑动选择器一般有水平和垂直两种，小程序中只提供了水平的形式，滑动到最左边是最小值，滑动到右边是最大值。`<slider/>`属性如下：

- `min`: 最小值，默认值为0。
- `max`: 最大值，默认值为100。
- `step`: 步长，取值必须大于0，并且可被（`max-min`）整除，默认值为1。
- `disabled`: 是否禁用，默认值为false。
- `value`: 当前取值，默认值为0。`value`值应该在`max`和`min`的区间范围内，设置后滑块会滚动到对应位置。
- `color`: 背景条的颜色，默认值为#e9e9e9。
- `selected-color`: 已选择的颜色，默认值为#000000。
- `show-value`: 是否在右侧显示当前`value`。

·bindchange: 完成一次拖动后触发的事件，event.detail={value: value}。

本示例中我们创建两个滑动选择器，分别绑定change事件用于控制图标的大小和透明度，如图4-16所示。

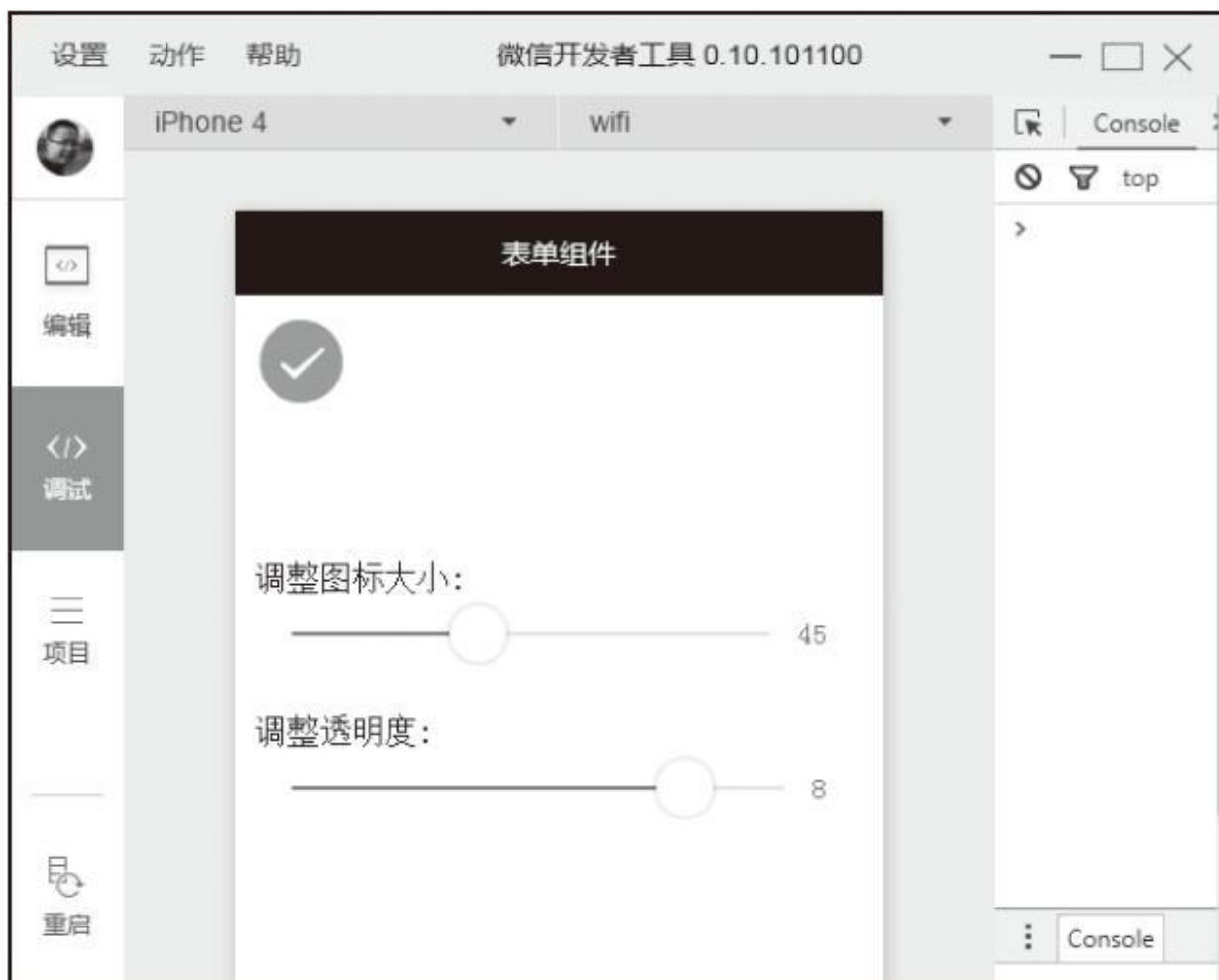


图4-16 slider示例

代码如下:

```

<view class="section icon-wrapper">
  <icon type="success" size="{{icon.size}}"
    style="opacity:{{icon.opacity/10}};"/>
</view>

<view class="section">
  <view>调整图标大小:</view>
  <slider show-value max="100" min="10" step="5" value="{{icon.size}}"
    bindchange="changeSize"></slider>
</view>

<view class="section">
  <view>调整透明度:</view>
  <slider show-value max="10" min="0" step="1" value="{{icon.opacity}}"
    bindchange="changeOpacity"></slider>
</view>

.section { padding : 10px; }
.section.icon-wrapper { height : 100px; font-size : 12px; }

Page( {
  data : {
    icon : {
      size : 20,
      opacity : 8
    }
  },
  /* 改变大小 */
  changeSize : function( e ) {
    this.data.icon.size = e.detail.value;
    this.setData( this.data );
  },
  /* 改变透明度 */
  changeOpacity : function( e ) {
    this.data.icon.opacity = e.detail.value;
    this.setData( this.data );
  }
} );

```

滑动选择器在移动端交互体验优于字符输入，在项目中一些数值输入项都可以考虑使用滑动选择器代替。

4.4.6 picker组件

`<picker/>`可以在屏幕底部弹出一个窗口，供用户在所提供的选择项中选择一个。`<picker/>`本身不会向用户呈现任何特殊效果，像`<checkbox-group/>`一样用于包裹其他组件，点击`<picker/>`包裹内的元素时会从底部弹出相应选项。`<picker/>`分为3种类型：普通选择器、时间选择器和日期选择器，默认是普通选择器，这三种选择器在细节上略有不同，我们可以通过设置`<picker/>`组件`mode`属性值切换不同选择器。

1. 普通选择器

普通选择器是默认的滚动选择器，我们只需要绑定数组类型的数据就能直接使用，对应的`mode`属性值为`selector`，普通选择器具有以下属性：

- `range`：底部弹出选项的数组，默认值为一个空数组`[]`。只有当的`mode`为`selector`时，`range`属性才有效。

- `rang-key`：当`range`是一个Object Array时，通过`rang-key`来指定Object中`key`的值作为选择器显示内容。

·value: mode为selector时，value值是数字，表示选择了range中的第几个，从0开始。

·bindchange: value改变时触发change事件，event.detail={value: value}。

·disabled: 是否禁用，默认值为false。

在下面的示例中我们创建了一个<view/>组件用于布局，点击后显示相应选项，如图4-17所示。

代码如下：

```
<picker value="{{selectedIndex}}" range="{{list}}" bindchange="change">
  <view class="picker">
    当前选中:{{list[selectedIndex]}}
  </view>
</picker>

.picker{ border:solid 1px #ddd; background-color: #fafafa; padding : 10px; }

Page( {
  data : {
    //创建对应选择项
    list : [
      '选项1',
      '选项2',
      '选项3'
    ],
    selectedIndex : 0
  },
  change : function( e ) {
    //修改选中项文案
    this.setData( {
      selectedIndex : e.detail.value
    } );
  }
} );
```

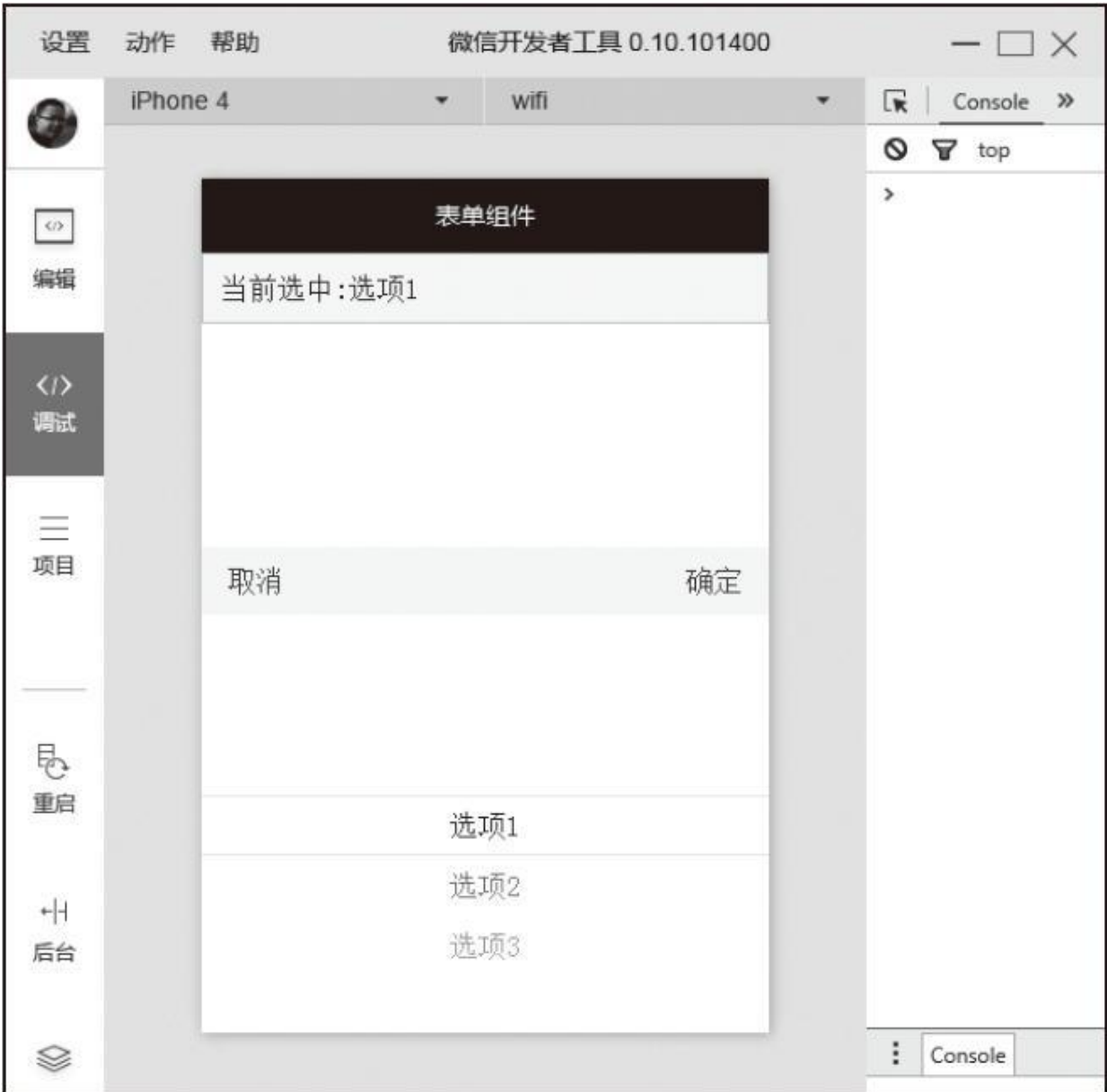


图4-17 普通选择器示例

2.时间选择器

在普通选择器基础上，`<picker/>`提供了时间选择器，对应的`mode`属性值为`time`，时间选择器具有以下属性：

·**value**: 表示选中的时间，字符串格式为“hh: mm”，默认为空。

·**start**: 表示有效时间范围的开始，字符串格式为“hh: mm”，默认值为空。

·**end**: 表示有效时间范围的结束，字符串格式为“hh: mm”，默认值为空。

·**bindchange**: value改变时触发change事件，event.detail={value: value}。

·**disabled**: 是否禁用，默认值为false。

下面举例说明，如图4-18所示。

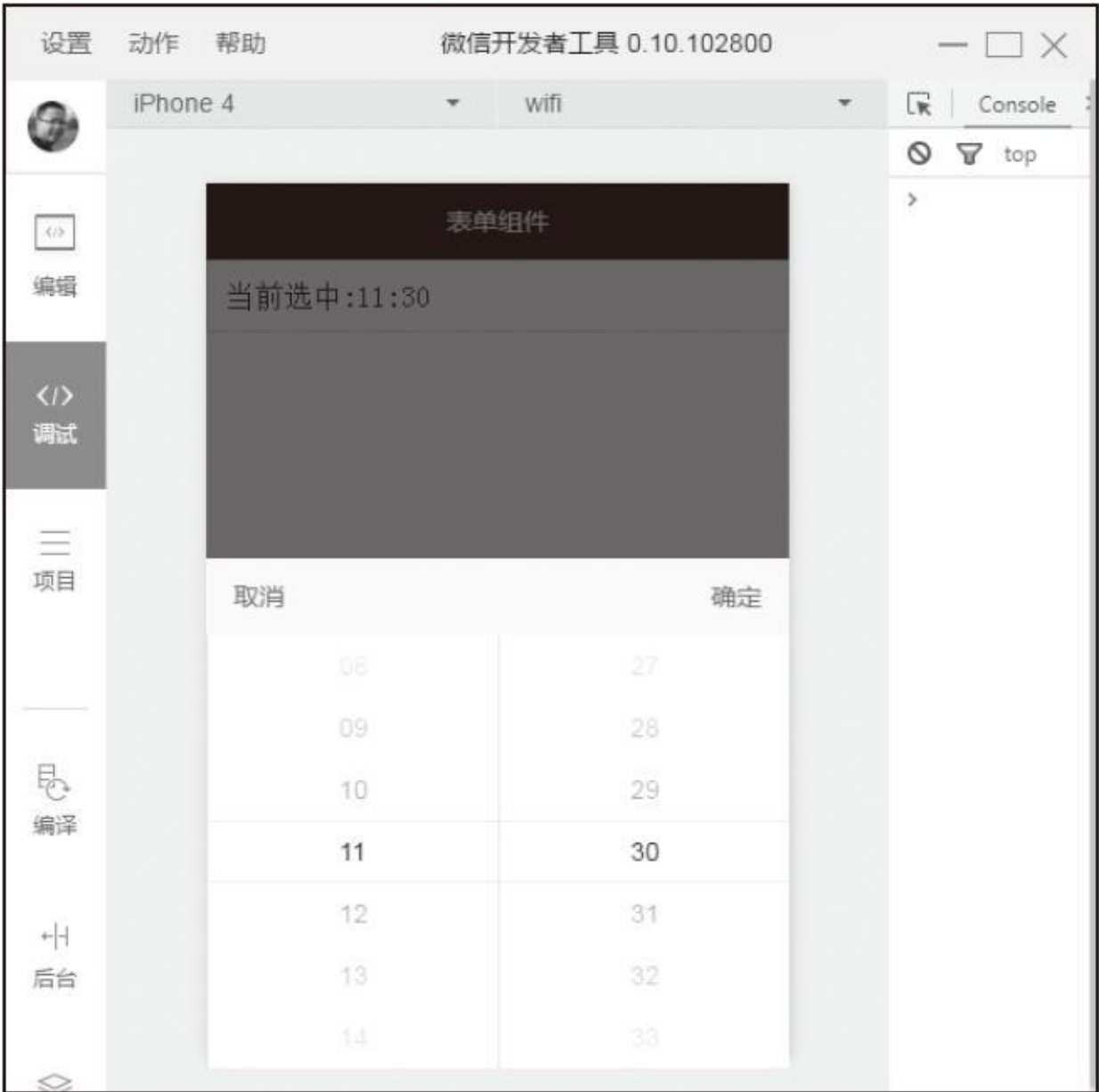


图4-18 时间选择器示例

代码如下:

```
<picker value="{{selectTime}}" mode="time" start="{{startTime}}"
  end="{{endTime}}" bindchange="change">
  <view class="picker">
    当前选中:{{selectTime}}
  </view>
</picker>
```



```
.picker{ border:solid 1px #ddd; padding : 10px; }  
Page( {  
  data : {  
    startTime : "00:00",  
    endTime : "24:00",  
    selectTime : "11:30"  
  },  
  //修改选中项文案  
  change : function( e ) {  
    this.setData( {  
      selectTime : e.detail.value  
    } );  
  }  
} );
```

3.日期选择器

日期选择器对应的mode属性为date，属性如下：

- value: 表示选中的日期，字符串格式为“yyyy-MM-dd”，默认值为0。
- start: 表示有效日期范围的开始，字符串格式为“yyyy-MM-dd”。
- end: 表示有效日期范围的结束，字符串格式为“yyyy-MM-dd”。
- fields: 表示选择器的粒度，有效值为year、month、day，默认值为day。
- bindchange: value改变时触发change事件，event.detail={value: value}。
- disabled: 是否禁用，默认值为false。

下面举例说明，如图4-19所示。

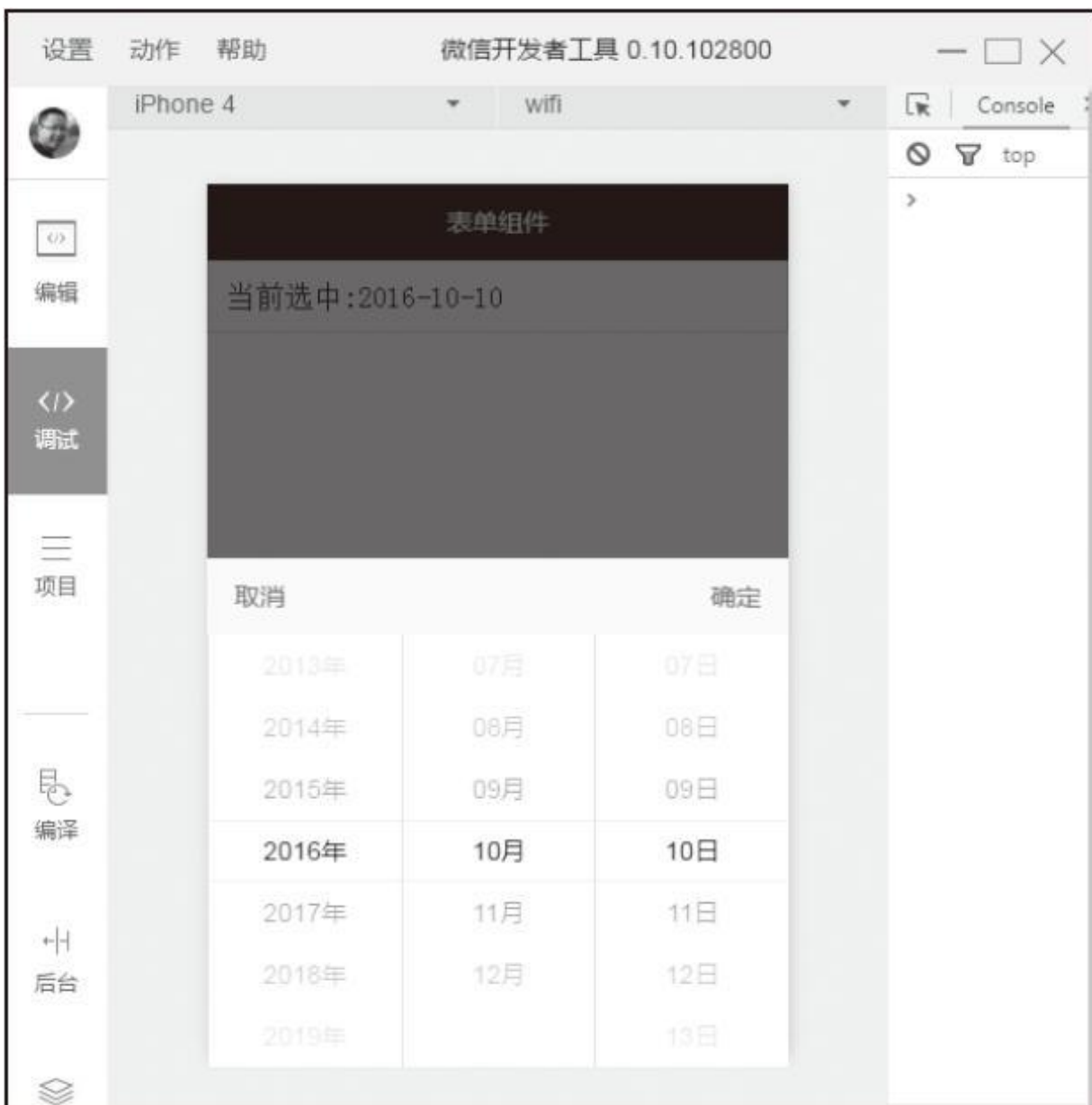


图4-19 日期选择器示例

代码如下:

```
<picker value="{{selectDate}}" mode="date" start="{{startDate}}"
        end="{{endDate}}" filed="day" bindchange="change">
  <view class="picker">
    当前选中:{{selectDate}}
  </view>
</picker>
```

```
.picker{ border:solid 1px #ddd; padding : 10px; }
```

```
Page( {  
  data : {  
    startDate : "2016-02-01",  
    endDate : "2016-12-30",  
    selectDate : "2016-10-10"  
  },  
  change : function( e ) {  
    this.setData( {  
      selectDate : e.detail.value  
    } );  
  }  
} );
```

4.4.7 picker-view组件

`<picker/>`一共提供了3类选择器，这3类选中器在模式、交互上都比较固定，而在业务场景中我们可能会涉及多种形态选择器，针对这种情况，小程序提供了`<picker-view/>`用于实现自定义滚动选择器，在`<picker-view/>`中我们能按自己意愿插入任意数量列，同时也能设置整个`<picker-view/>`样式。一个完整的`<picker-view/>`包含两个标签：

`<picker-view/>`和`<picker-view-column/>`，`<picker-view-column/>`用于创建列，列中孩子节点高度会自动设置为`<picker-view/>`的选中框高度，`<picker-view/>`中仅可放置`<picker-view-column/>`，放置其他节点不会显示。`<picker-view/>`属性如下：

- `value`：数组类型，数组中的数字依次表示`<picker-view/>`内的`<picker-view-colume/>`选择的第几项（下标从0开始），数字大于`<picker-view-column/>`可选项长度时，选择最后一项。

- `indicator-style`：设置选择器中间选中框的样式。

- `bingchange`：当滚动选择，`value`改变时触发`change`事件，`event.detail={value: value}`；`value`为数组，表示`picker-view`内的`picker-view-column`当前选择的是第几项（下标从0开始）。

下面用示例说明，如图4-20所示。本示例基于<picker-view/>实现一套电商系统常用的配送时间选择器，每个日期对应一套自己的配送时段，每次切换日期，对应的时段也会进行相应切换。



图4-20 自定义滚动选择器

代码如下：

```
<view class="custom-picker">
  <view class="title">
    自定义滑动选择器
  </view>
  <!-- 设置内部选中选项高度 -->
  <picker-view indicator-style="height:80px;" bindchange="changeTime">
```

```

<!-- 日期列 -->
<picker-view-column>
  <view class="cell" wx:for="{{list}}" wx:key="index">
    {{item.date}}
  </view>
</picker-view-column>
<!-- 时段列 -->
<picker-view-column>
  <block wx:for="{{list}}" wx:key="{{index}}" wx:if="{{item.selected}}">
    <view class="cell" wx:for="{{item.times}}" wx:for-index="subIndex"
      wx:for-item="subItem" wx:key="subIndex">
      {{subItem.time}}
    </view>
  </block>
</picker-view-column>
</picker-view>
</view>

.custom-picker { top : 100px;width :80%; margin: 0 auto; margin-top : 200rpx;
border:solid 1px #ccc; }
.custom-picker .title { text-align:center;color:#444; padding : 20rpx;
font-size:24rpx; background-color:#eee;border-bottom:solid 1px #ccc }
.custom-picker picker-view{ height:200px; }
.custom-picker picker-view .cell { text-align: center; line-height: 80px; }
.custom-picker picker-view-column { border-right:solid 1px #ccc; position:
relative; left : 1px; }

Page( {
  data : {
    list : [{
      date : '12月27日',
      selected : true,
      times : [{time : '19:00'},{time : '19:30'},{time : '20:00'},
        {time : '20:30'},{time : '21:00'}]
    },{
      date : '12月28日',
      selected : false,
      times : [{time : '9:00'},{time : '9:30'},{time : '10:00'},
        {time : '10:30'},{time : '11:00'}]
    },{
      date : '12月29日',
      selected : false,
      times : [{time : '12:00'},{time : '12:30'},{time : '13:00'},
        {time : '14:30'},{time : '15:00'}]
    }
  ]
},
// 日期改变时渲染对应时间
changeTime : function( e ) {
  var index = e.detail.value[0], // 只监控日期改变，时间改变忽略
    list = this.data.list,
    i, d;

  for ( i = 0; d = list[i]; ++i ) {
    d.selected = i == index ? true : false;
  }
  this.setData( this.data )
}
} );

```

4.4.8 input组件

<input/>是单行输入框，用于收集用户信息。根据不同的type属性值，输入字段拥有很多形式，与HTML不同的是小程序中的<input/>类型没有按钮类型，都是与输入相关的类型。<input/>属性较多，但大部分都和HTML的<input/>相似，其属性如下：

- value: 输入框的初始内容。
- type: input的类型，有效值为：text、number、idcard、digit、time、date。
- password: 是否显示密码类型，默认为false。
- placeholder: 输入框为空时的占位符。
- placeholder-style: 指定placeholder的样式。
- placeholder-class: 指定placeholder的样式类，默认值为input-placeholder。
- disabled: 是否禁用，默认为false。
- maxlength: 最大输入长度，设置为0的时候不限制最大长度，默认值为140。

·**cursor-spacing**: 指定光标与键盘的距离，单位px。取input距离底部的距离和**cursor-spacing**指定的距离的最小值作为光标与键盘的距离。

·**auto-focus**: 自动聚焦，拉起键盘（开发工具暂不支持）。页面中只能有一个<input/>或<textarea/>设置**auto-focus**属性。

·**focus**: 获取焦点（开发工具暂不支持），默认为**false**。

·**bindinput**: 当键盘输入时，触发input事件，**event.detail={value: value}**，处理函数可以直接**return**一个字符串，将替换输入框的内容。

·**bindfocus**: 输入框聚焦时触发，**event.detail={value: value}**。

·**bindblur**: 输入框失去焦点时触发，**event.detail={value: value}**。

·**bindconfirm**: 点击完成按钮时触发，**event.detail={value: value}**。

下面举例说明（如图4-21）input组件的使用。本例中对光标位置的处理需要大家多留意，其他大部分属性大家应该都比较熟悉，这里就不一一举例。



图4-21 input示例

代码如下：

```
<view class="section">
  <input placeholder="默认样式，自动聚焦" auto-focus/>
</view>

<view class="section">
  <input placeholder="内容中123会被替换为0" bindinput="changeValue"
    type="number" maxlength="20"/>
</view>

<view class="section">
  <input placeholder="输入3个以上字符会收起键盘" bindinput="hideKeyboard"
    type="number"/>
</view>

.section { font-size : 12px; padding : 10px 5px; border-bottom: dashed 1px
#cecece; }
.section input { border : solid 1px #ccc; padding : 0 5px; background-color: #fff;
```

```
border-radius: 4px; height : 30px; }

Page( {
  changeValue : function( e ) {
    console.log( e.detail );
    var value = e.detail.value,
        pos = e.detail.cursor,
        left;

    // 计算光标位置
    if ( pos !== -1 ) {
      // 光标在中间位置
      left = value.slice( 0, pos );
      // 修改后光标位置要随之变化
      pos = left.replace( /123/g, '2' ).length;
    }
    return {
      value : e.detail.value.replace( /123/g, '2' ),
      cursor : pos
    }
  },
  hideKeyboard : function( e ) {
    if ( e.detail.value.length >= 3 ) {
      //调用关闭键盘API
      wx.hideKeyboard();
    }
  }
} );
```

4.4.9 textarea组件

`<textarea/>`是多行输入框，与HTML中多行输入框不一样的是，小程序中`<textarea/>`是一个自闭合标签，它的值需要赋值给`value`属性，而不是被标签包裹。与`<input/>`相比，大部分属性都一样。其属性如下：

- `value`：输入框的初始内容。
- `placeholder`：输入框为空时的占位符。
- `placeholder-style`：指定`placeholder`的样式。
- `placeholder-class`：指定`placeholder`的样式类，默认值为`textarea-placeholder`。
- `disabled`：是否禁用，默认为`false`。
- `maxlength`：最大输入长度，设置为0的时候不限制最大长度，默认值为140。
- `auto-focus`：自动聚焦，拉起键盘（开发工具暂不支持）。页面中只能有一个`<input/>`或`<textarea/>`设置`auto-focus`属性。
- `focus`：获取焦点（开发工具暂不支持），默认为`false`。

·**auto-height**: 是否自动增高，设置**auto-height**时，**style.height**不生效，初始状态默认为一行高度。

·**fixed**: 如果**textarea**是在一个**position: fixed**的区域，需要显示指定属性**fixed**为**true**，默认值为**false**。

·**cursor-spacing**: 指定光标与键盘的距离，单位**px**。取**textarea**距离底部的距离和**cursor-spacing**指定的距离的最小值作为光标与键盘的距离。默认值为0。

·**bindinput**: 当键盘输入时，触发**input**事件，**event.detail={value: value}**，**bindinput**处理函数的返回值并不会反映到**textarea**上。

·**bindinput**: 除了**date/time**类型外的输入框，当键盘输入时，触发**input**事件，**event.detail={value: value, cursor: 光标位置}**，处理函数可以直接**return**一个字符串，替换输入框的内容。

·**bindfocus**: 输入框聚焦时触发，**event.detail={value: value}**。

·**bindblur**: 输入框失去焦点时触发，**event.detail={value: value}**。

·**bindlinechange**: 输入框行数变化时调用，首次渲染时也会触发，**event.detail={height: 0, heightRpx: 0, lineCount: 0}**;

·bindconfirm: 点击完成时，触发confirm事件，event.detail={value: value}。

下面用示例说明，这个示例创建了一个<textarea/>，并在失焦时打印出当前输入内容，如图4-22所示。

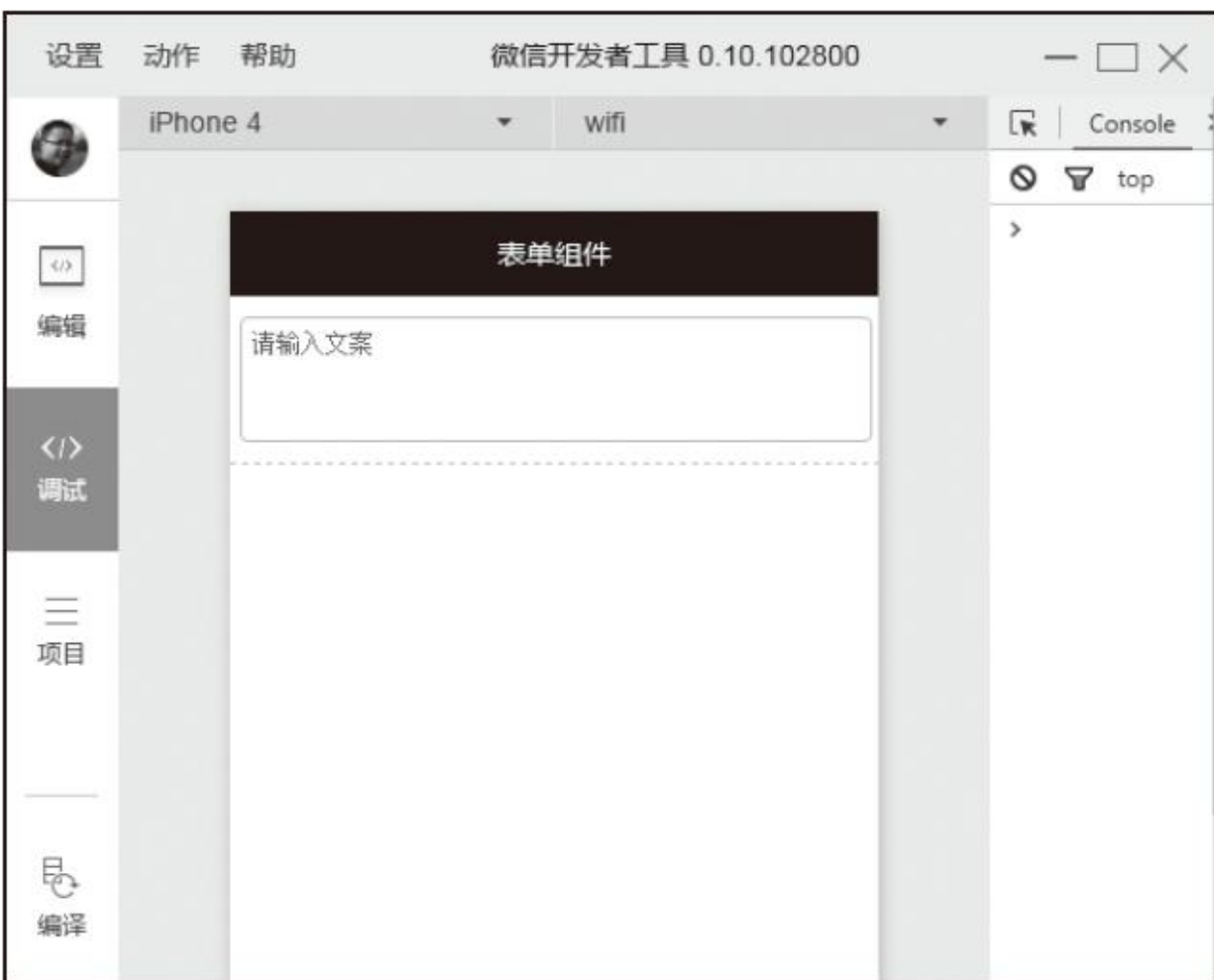


图4-22 textarea示例

代码如下：

```
<view class="section">
  <textarea bindblur="getValue" placeholder="请输入文案"
    placeholder-class="textarea-holder"/>
</view>

.section { font-size : 12px; padding : 10px 5px; border-bottom: dashed 1px
#cecece; }
.section textarea { border : solid 1px #ccc; padding : 5px; background-color:
#fff; border-radius: 4px; height : 50px; }
.section .textarea-holder { color : red; }

Page( {
  getValue : function( e ) {
    // 由于没有bindinput可以用blur代替
    console.log( e.detail.value );
  }
} );
```

使用<textarea/>时需要注意以下几点:

- 微信6.3.30中，<textarea/>在列表渲染时，新增加的<textarea/>在自动聚焦时的位置计算错误。

- 请勿在scroll-view中使用<textarea/>。

- <textarea/>的blur事件会晚于页面上的tap事件，如果需要在button的点击事件获取<textarea/>，可以使用<form/>的bindsubmit。

- 不建议在多行文本上对用户的输入进行修改，所以<textarea/>的bindinput处理函数并不会将返回值反映到<textarea/>上。

4.4.10 button组件

按钮除了在中应用中提供交互功能，按钮的颜色也承载了应用的引导作用，通常在一个程序中一个按钮至少有3种状态：默认点击

（**default**）、建议点击（**primary**）、谨慎点击（**warn**）。在构建项目时，注意在合适的场景使用合适的按钮。当`<button/>`被`<form/>`包裹时，可以通过设置**form-type**属性触发表单对应的事件，`<button/>`组件除了系统定制的几种类型，也可以通过在标签中嵌套其他组件实现自定义按钮，其属性如下：

- size**: 表示按钮的大小，有效值为**default**、**mini**，默认值为**default**。

- type**: 按钮的样式类型，有效值为**primary**、**default**、**warn**，默认值为**default**。

- plain**: 按钮是否镂空，背景色透明，默认值为**false**。

- disabled**: 是否禁用，默认值是**false**。

- loading**: 名称前是否带**loading**图标，默认值为**false**。通常在表单提交过程中或者按钮点击后等待反馈时，就需要打开**loading**让用户有感知。

·**form-type**: 用于<form/>组件，有效值为submit、reset，点击一个位于<form/>中且设置了form-type的<button/>时，会触发<form/>的submit事件或reset事件。

·**hover-class**: 指定按钮按下去的样式类。当hover-class="none"时，没有点击态效果，默认值为button-hover。button-hover的默认样式为{background-color: rgba (0, 0, 0, 0.1) ; opacity: 0.7}，如果想进行系统级修改可以直接覆盖这个样式。

·**hover-start-time**: 按住后多久出现点击态，单位毫秒，默认值50。

·**hover-stay-time**: 手指松开后点击态保留时间，单位毫秒，默认值400。

button大部分属性都和样式有关，一些属性外的特殊样式大家也可以通过设置wxss来实现。

我们在下面的示例中为大家列举了按钮不同属性下的UI状态，如图4-23所示，form-type属性将在<form/>章节的示例中进行演示：

代码如下：

```
<button>默认按钮样式</button>
<button size="mini">size:mini</button>
```



图4-23 button示例

```
<button type="primary">type:primary</button>
<button type="warn">type:warn</button>
<button plain>plain</button>
<button disabled>disabled</button>
<button loading>loading</button>
<!-- 按钮按下去后会变成红色 -->
<button hover-class="my-button-hover">hover-class:my-button-hover</button>
<!-- 嵌套多媒体的自定义按钮 -->
<button>
  
</button>

button { margin : 5px 0; }
.my-button-hover { background-color: red; color : #fff; }
```

4.4.11 form组件

`<form/>`是本章最后一个也是本章最关键的一个组件，它用于嵌套本章其他组件，使之形成表单。当触发`<form/>submit`方法时，`<form/>`能将组件内用户输入的`<switch/>`、`<input/>`、`<checkbox/>`、`<slider/>`、`<radio/>`、`<picker/>`数据按组件`name`属性进行组装，作为参数传递给`submit`方法。通过这种方式，我们可以利用`<form/>`很方便地获取表单数据同后台进行交互。`<form/>`有以下属性：

- `report-submit`：是否返回`formId`，`formId`用于发送模板消息，默认值为`false`。

- `bindsubmit`：携带`form`中的数据触发的`submit`事件，与`<button/>`的`form-type`属性配合使用，`event.detail={value: {'name': 'value'}, formId: ''}`。

- `bindreset`：表单重置时触发`reset`事件，与`<button/>`的`form-type`属性配合使用。

下面的示例构建了一个简单的表单，点击提交按钮后，控制台会打印出用户输入项，如图4-24所示。

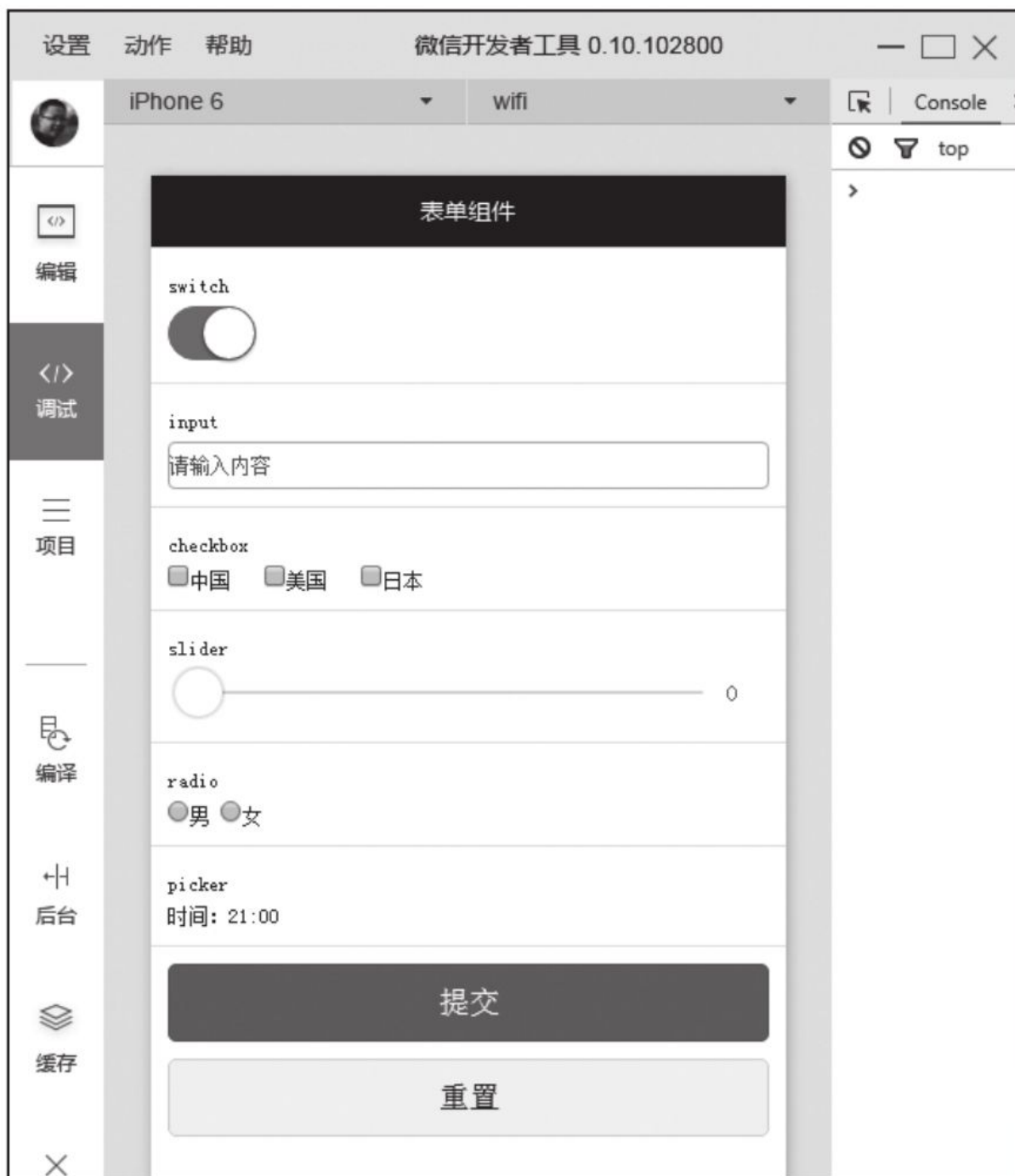


图4-24 form示例

代码如下：

```

<form bindsubmit="submit" bindreset="reset">
  <view class="section">
    <view class="title">switch</view>
    <switch name="myswitch"/>
  </view>

  <view class="section">
    <view class="title">input</view>
    <input name="myinput" placeholder="请输入内容" value="{{textvalue}}"/>
  </view>

  <view class="section">
    <view class="title">checkbox</view>
    <checkbox-group name="mycheckbox">
      <label><checkbox value="1"/>中国</label>
      <label><checkbox value="2"/>美国</label>
      <label><checkbox value="3"/>日本</label>
    </checkbox-group>
  </view>

  <view class="section">
    <view class="title">slider</view>
    <slider name="myslider" show-value/>
  </view>

  <view class="section">
    <view class="title">radio</view>
    <radio-group name="myradio">
      <label>
        <radio value="0"/>男
        <radio value="1"/>女
      </label>
    </radio-group>
  </view>

  <view class="section">
    <view class="title">picker</view>
    <picker value="{{index}}" range="{{times}}" bindchange="changePicker">
      <view>
        时间: {{times[index]}}
      </view>
    </picker>
  </view>

  <view class="section">
    <button type="primary" form-type="submit">提交</button>
    <button form-type="reset">重置</button>
  </view>
</form>

```

```

.section { font-size: 12px; padding : 10px; border-top : solid 1px #eee; }
.section input { border : solid 1px #ccc; border-radius: 4px; }
.section button { margin-bottom : 10px; }
.section label { margin-right : 20px; }
.section .title{ margin: 5px 0; }

```

```

Page( {
  data : {
    times : [
      '20:00',
      '20:30',
      '21:00',
      '21:30',
      '22:00'
    ],
    index : 3
  },

```

```
//时间选择器切换时，修改对应值
changePicker : function( e ) {
    this.setData( {
        index : e.detail.value
    } );
},
//点击提交时打印当前输入数据
submit : function( e ) {
    console.log( e.detail.value );
},
//重置表单
reset : function( e ) {
    console.log( '已经重置对象' )
}
} );
```

4.5 导航组件

`<navigator/>`是小程序中的页面链接，其作用和HTML中超链接标签一样，主要控制页面的跳转。链接内容可以是一个字、一个词或者一组词，也可以是一幅图，你可以点击这些内容来跳转到新的页面，`<navigator/>`属性如下：

- `url`：应用内跳转链接。链接地址为需要跳转页面的相对地址。
- `redirect`：跳转行为是否为重定向，如果为`true`，则跳转时会关闭当前页面。默认值为`false`。
- `open-type`：可选值'`navigate`'、'`redirect`'、'`switchTab`'，对应于`wx.navigateTo`、`wx.redirect-To`、`wx.switchTab`的功能，默认值为`navigate`。
- `hover-class`：点击时的样式类，当`hover-class="none"`时，没有点击效果。默认值为`navigator-hover`。`navigator-hover`默认为`{background-color: rgba (0, 0, 0, 0.1) ; opacity: 0.7; }`，`<navigator/>`子节点背景色应为透明色。

页面间跳转可以通过`url`进行参数传递，规则符合URL协议，新页面可以通过注册`onLoad`方法获取参数，我们也可以通过`redirect`和`url`的

配合实现刷新当前页面，如图4-25所示。

navigator页面代码如下：

```
<navigator url="../../recevie/recevie?name=weixinapp&time=2016">  
    跳转到其他页面  
</navigator>  
<navigator url="../../recevie/recevie?name=weixinapp" hover-class="myhover"  
    redirect>  
    重定向到其他页面  
</navigator>  
<navigator url="../../navigator/navigator?name=weixinapp" hover-class="myhover"  
    redirect>  
    刷新当前页面  
</navigator>
```

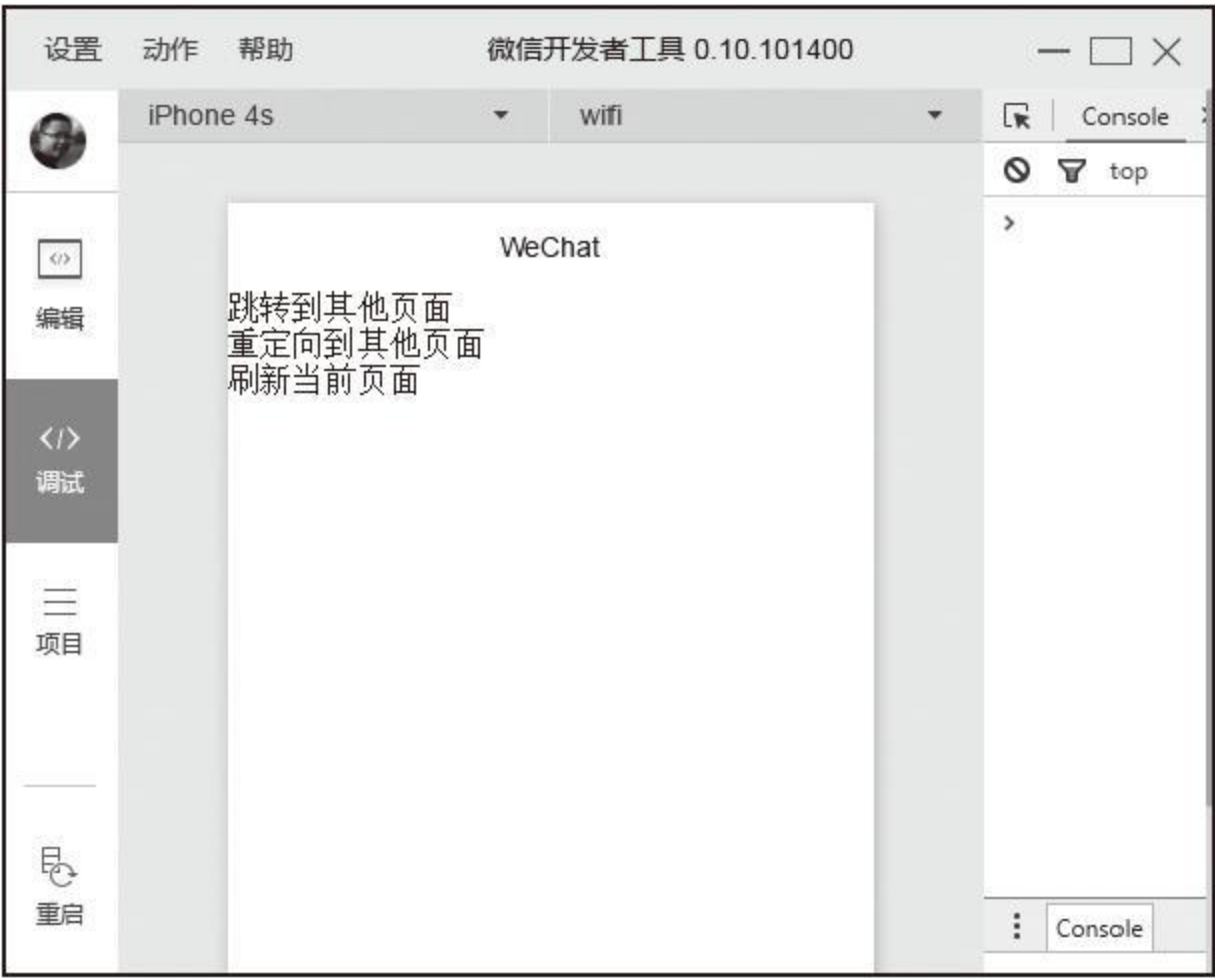


图4-25 navigator页面

我们在recevie页面通过注册onLoad事件获取传入的参数，如图4-26所示。

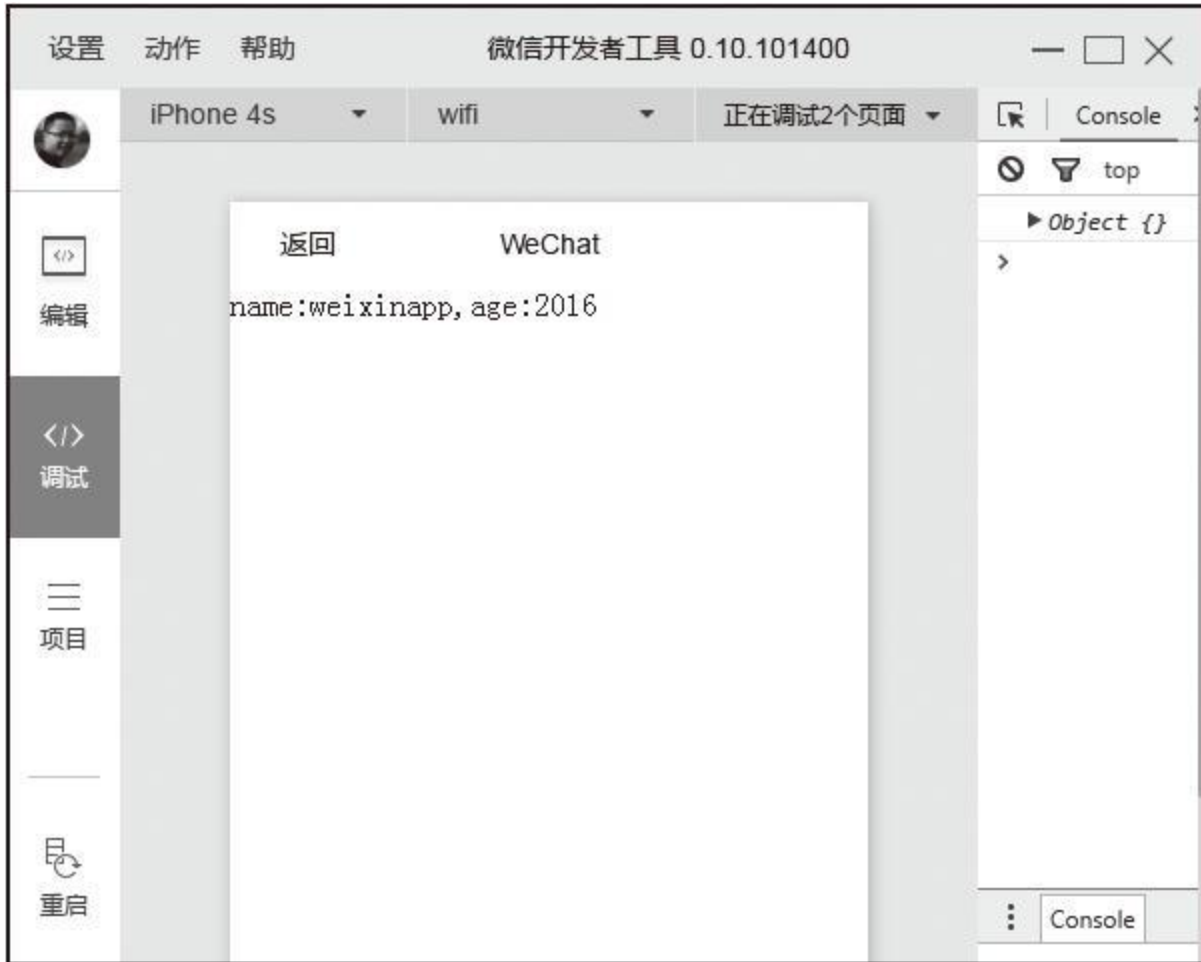


图4-26 receive页面

recevie页面如下：

```
<view>
  name:{{query.name}},age:{{query.time}}
</view>

Page( {
  data : {
    query : {}
  },
  onLoad : function( query ) {
    /* 框架已将参数转化为onLoad参数 */
```

```
        this.data.query = query;  
        this.setData( this.data );  
    }  
} );
```

一些简单的数据可以通过url进行传递，但一些较为复杂的数据对象就需要我们通过架构实现。

4.6 媒体组件

4.6.1 image

一个应用中图片是必不可少的，就像HTML提供了标签一样，小程序提供了<image/>组件。小程序中的<image/>除了可以显示图片外，还提供了图片的裁剪、缩放模式属性，这大大丰富了<image/>的显示能力。<image/>默认宽度为300px，默认高度为225px，属性如下：

- src: 图片资源地址。可以是网络地址，也可以是本地图片的相对地址。

- mode: 图片的裁剪、缩放模式。默认值为“scaleToFill”。

- binderror: 当错误发生时，发布到App Service的事件名，事件对象event.detail={errMsg: 'something wrong'}。

- bindload: 当图片载入完毕时，发布到App Service的事件名，事件对象event.detail={height: '图片高度px', width: '图片宽度px'}。

- mode属性有一共有13种裁剪模式，其中4种是缩放模式，9种是裁剪模式。

·缩放模式:

·**scaleToFill**: 不保持纵横比缩放图片, 使图片的宽高完全拉伸至填满image元素。

·**aspectFit**: 保持纵横比缩放图片, 使图片的长边能完全显示出来。也就是说, 可以完整地将图片显示出来。

·**aspectFill**: 保持纵横比缩放图片, 只保证图片的短边能完全显示出来。也就是说, 图片通常只在水平或垂直方向是完整的, 另一个方向将会发生截取。

·**widthFix**: 宽度不变, 高度自动变化, 保持原图宽高不变, 这时图片原有高度样式会失效。

·裁剪模式:

·**top**: 不缩放图片, 只显示图片的顶部区域。

·**bottom**: 不缩放图片, 只显示图片的底部区域。

·**center**: 不缩放图片, 只显示图片的中间区域。

·**left**: 不缩放图片, 只显示图片的左边区域。

·**right**: 不缩放图片, 只显示图片的右边区域。

- top left: 不缩放图片，只显示图片的左上边区域。

- top right: 不缩放图片，只显示图片的右上边区域。

- bottom left: 不缩放图片，只显示图片的左下边区域。

- bottom right: 不缩放图片，只显示图片的右下边区域。

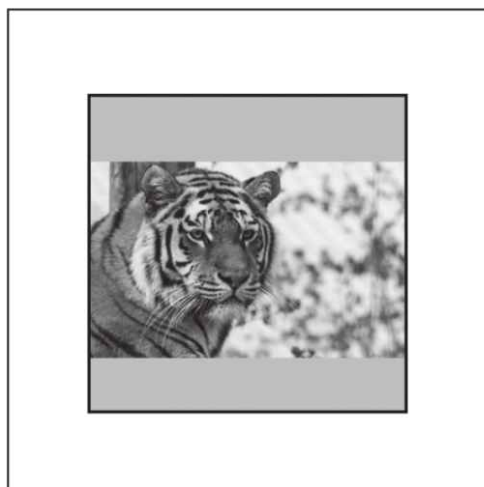
图4-27使用一张原图大小为720px×450px的图，和大小为300px×300px的image组件为大家展示不同模式下的显示效果。



a) 原图



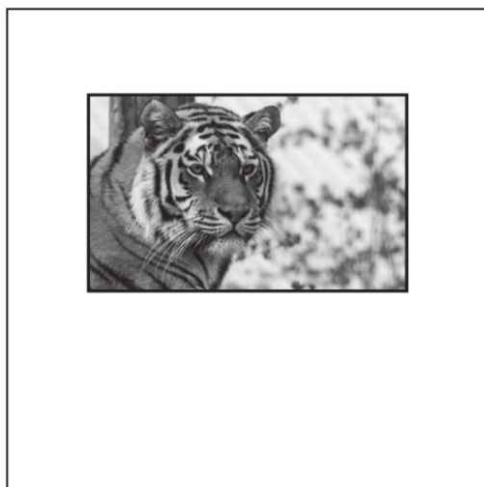
b) scaleToFill



c) aspectFit

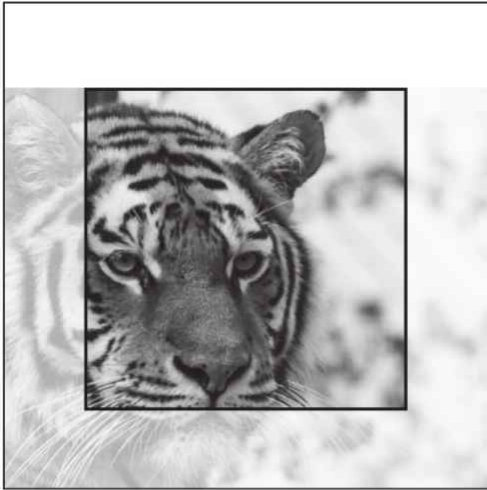


d) aspectFill

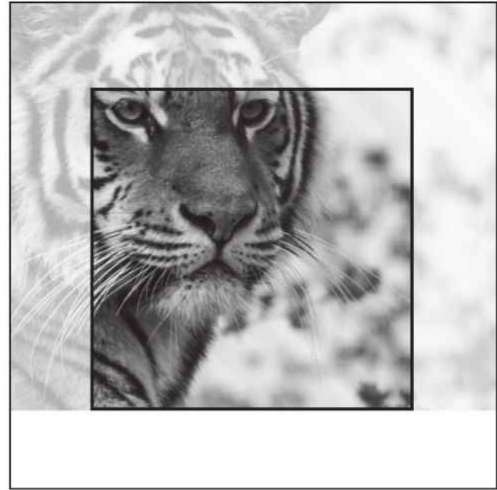


e) idthFix

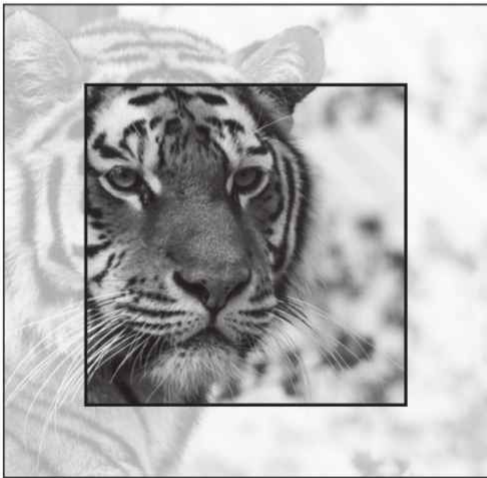
图4-27 image组件不同模式的效果图



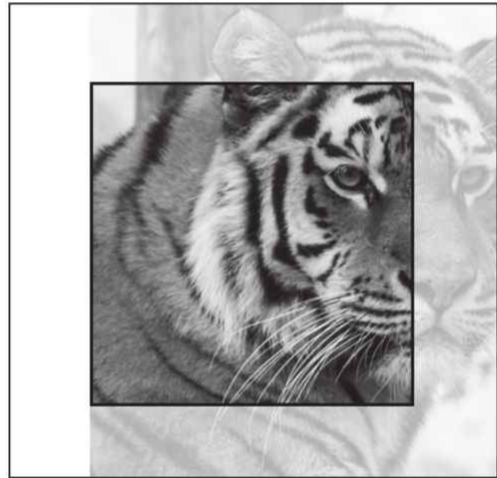
f) op



g) ottom



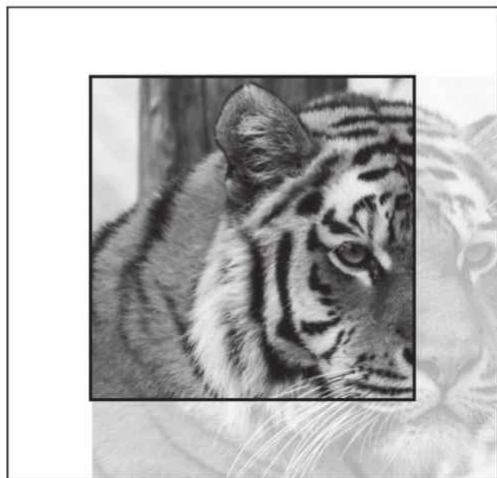
h) center



i) left

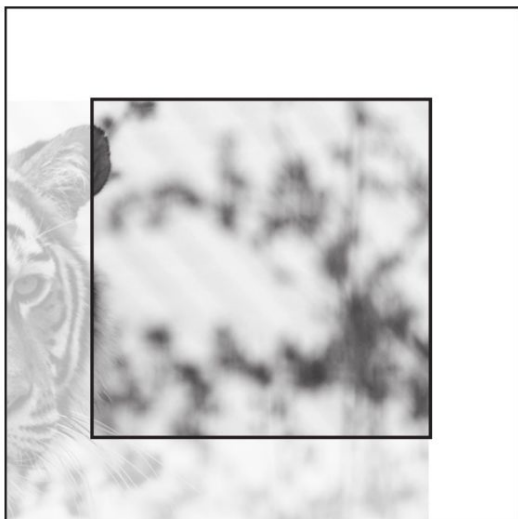


j) right

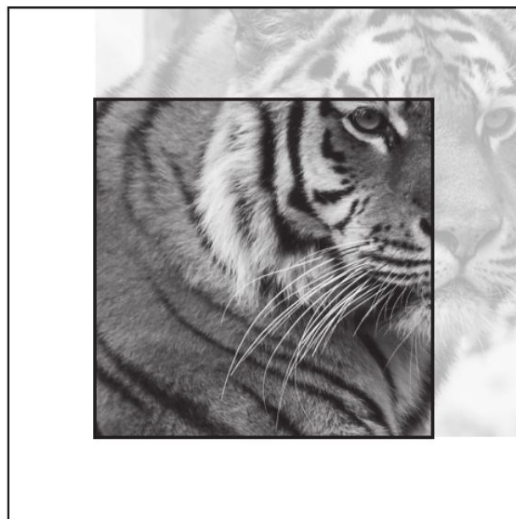


k) top left

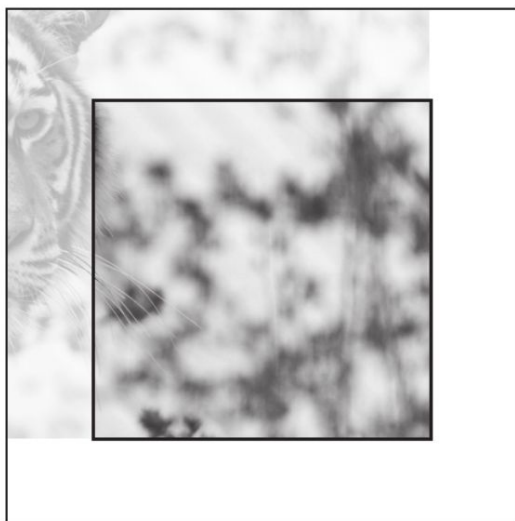
图4-27 (续)



l) top right



m) bottom left



n) bottom right

图4-27 (续)

4.6.2 audio

小程序运行在页面中直接嵌入的音频组件，官方系统为其提供了一套默认的组件样式，我们也可以通过修改属性或调用API封装自己的音频UI组件，<audio/>属性如下：

- src: 要播放音频的资源地址。可以是网络音频地址或本地音频文件相对路径。

- loop: 是否循环播放，默认值为false。

- controls: 是否显示默认控件，默认值为true。

- poster: 默认控件上的音频封面的图片资源地址，如果controls属性值为false则poster无效。

- name: 默认控件上的音频名字，如果controls属性值为false则name无效。默认值为“未知音频”。

- author: 默认控件上的作者名字，如果controls属性值为false则author无效。默认值为“未知作者”。

- binderror: 当发生错误时触发error事件，detail={errMsg: MediaError.code}。MediaError.code有以下类型：

- `MEDIA_ERR_ABORTED`: 获取资源被用户禁止。
- `MEDIA_ERR_NETWORK`: 网络错误。
- `MEDIA_ERR_DECODE`: 解码错误。
- `MEDIA_ERR_SRC_NOT_SUPPORTED`: 不合适资源。
- `bindplay`: 当开始/继续播放时触发`play`事件。
- `bindpause`: 当暂停播放时触发`pause`事件。
- `bindtimeupdate`: 当播放进度改变时触发`timeupdate`事件, `detail={currentTime, duration}`。
- `bindended`: 当播放到末尾时触发`ended`事件。

默认的`<audio/>`样式就是我们在微信朋友圈中常见的样式, 如图4-28所示。`<audio/>`播放等功能需要配合API实现, 具体可参考API音乐播放章节。

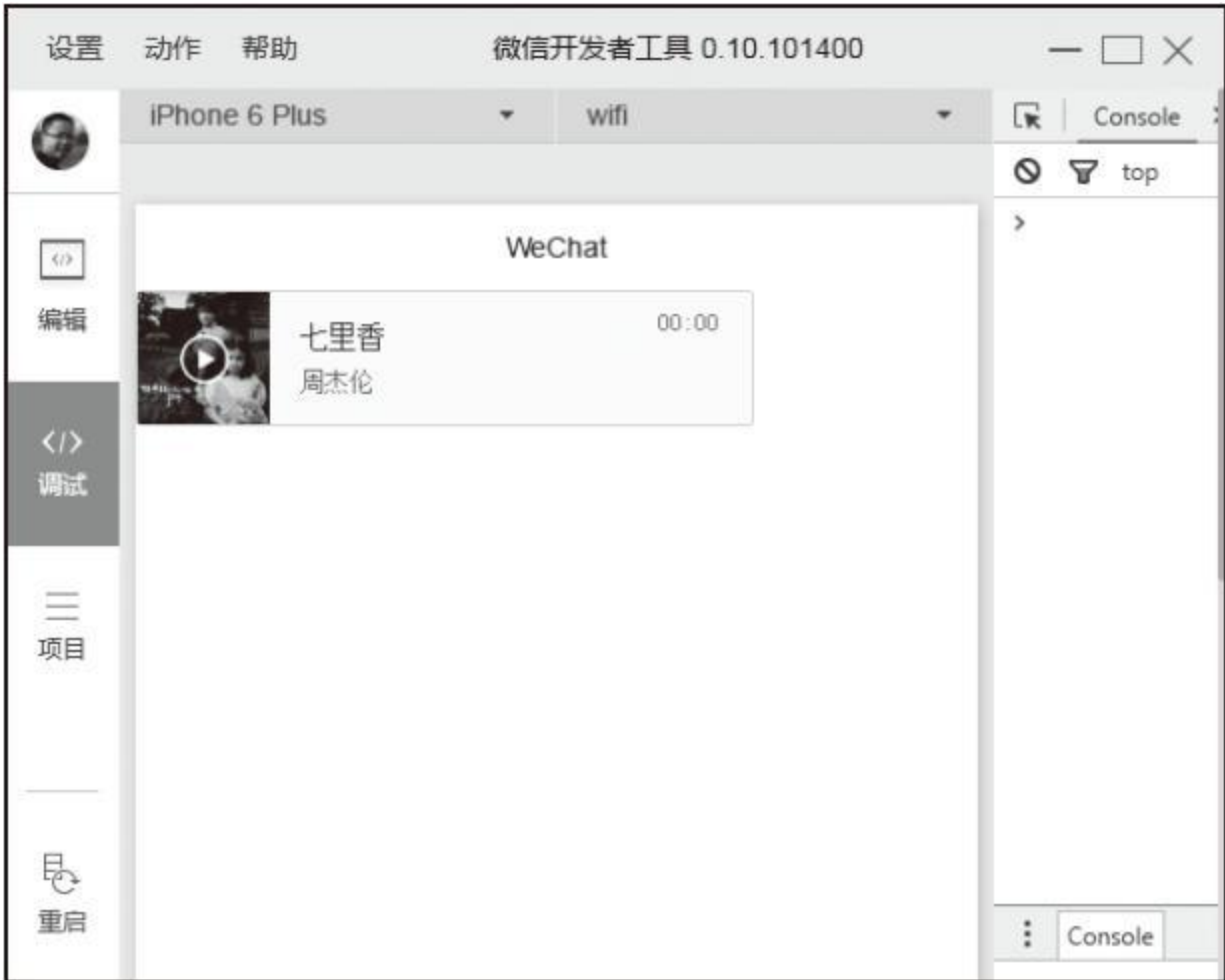


图4-28 系统默认控件

代码如下:

```
<audio poster="{{myaudio.poster}}" src="{{myaudio.src}}"
      name="{{myaudio.name}}" author="{{myaudio.author}}"
      controls loop action="{{myaudio.action}}">
</audio>

Page( {
  data : {
    myaudio : {
      src : './resource/qilixiang.mp3',
      poster :
'http://www.yoka.com/dna/pics/ba1c1117/42/d3551ec1c17cadcdcd1.jpg',
      name : '七里香',
      author : '周杰伦'
    }
  }
} );
```


4.6.3 video

小程序中允许我们简单嵌入视频，和audio组件相比，它能提供的属性少了很多，只能设置视频源，监听加载错误，这样几乎不能对video组件做什么操作。<video/>默认宽度为300px，高度为225px，属性如下：

- src: 要播放视频的资源地址。

- controls: 是否显示默认播放控件（播放/暂停按钮、播放进度、时间），默认为true。

- danmu-list: 弹幕列表。

- danmu-btn: 是否显示弹幕按钮，只在初始化时有效，不能动态变更，默认为false。

- enable-danmu: 是否展示弹幕，只在初始化时有效，不能动态变更，默认为false。

- autoplay: 是否自动播放，默认为false。

- bindplay: 当开始/继续播放时触发play事件。

·bindpuase: 当暂停播放时触发pause事件。

·bindended: 当播放到末尾时触发ended事件。

·bindtimeupdate: 播放进度变化时触发, event.detail={currentTime: '当前播放时间'}。触发频率应该在250ms一次。

·objectFit: 当视频大小与video容器大小不一致时, 视频的表现形式。contain: 包含, fill: 填充, cover: 覆盖, 默认值为contain。

vicle示例如图4-29的所示。



图4-29 video示例

代码如下：

```
<view class="video">
  <video id="myVideo" src="{{src}}" danmu-list="{{danmu}}" enable-danmu
controls></video>
  <view class="action">
    <button bindtap="getVideo">获取视频</button>
    <view class="danmu">
      <input type="text" value="{{danmuText}}" bindblur="setInputValue"/>
      <button bindtap="sendDanmu">发送弹幕</button>
    </view>
  </view>
</view>
```



```

.video video { width : 100%;height : 562.5rpx; }
.video .action { padding : 20rpx; }
.video .action .danmu { margin-top: 10px; position: relative; padding-right :
210rpx; height : 80rpx; }
.video .action .danmu input { border : solid 1px #ccc; height : 80rpx; padding :
0 10rpx; border-radius: 3px;}
.video .action .danmu button { width : 200rpx; position: absolute; right : 0;
bottom : 0; height : 80rpx; }

Page( {
  data : {
    src : 'http://wxsnsdy.tc.qq.com/105/20210/snsdyvideodownload?
filekey=30280201010421301f0201690402534804102ca905ce620b1241b726bc41dcff44e002040
12882540400&bizid=1023&hy=SH&fileparam=302c020101042530230204136ffd93020457e3c4ff
02024ef202031e8d7f02030f42400204045a320a0201000400',
    //设置弹幕
    danmu : [{
      text : '第1s出现的弹幕',
      color : '#ff0000',
      time : 1
    }, {
      text : '第2s出现的弹幕',
      color : '#00ff00',
      time : 2
    }
  ],
  danmuText : ''
},
onReady : function() {
  //获取video上下文videoContext对象
  this.videoContext = wx.createVideoContext( 'myVideo' );
},
getVideo : function() {
  var self = this;
  //从本地或相机选择视频
  wx.chooseVideo( {
    success : function( res ) {
      self.setData( {
        maxDuration : 60,
        src : res.tempFilePath
      } );
    }
  } );
},
setInputValue : function( e ) {
  // 同步数据
  this.setData( {
    danmuText : e.detail.value
  } );
},
//发送弹幕
sendDanmu : function() {
  var danmuText = this.data.danmuText;
  console.log( this.videoContext );
  this.videoContext.sendDanmu( {
    text : danmuText,
    color : '#ff0000'
  } );
  // 发送弹幕后清空输入框
  this.setData( {
    danmuText : ''
  } );
}
} );

```

需要注意的是`<video/>`组件不能在`<scroll-view/>`中使用。

4.7 地图组件

移动应用中地图是必不可少的内容，通过地图我们可以很直观地表现出地理信息，为此小程序提供了<map/>组件和定位相关的API。本小节主要讲解<map/>组件，它主要负责展示性的功能，地理位置的获取将在后续API中讲解。由于需要适配三端，在小程序中我们不能使用<map/>组件以外的地图插件，如高德地图、百度地图，经过几次发版后，目前小程序地图组件正能满足大部分需求。

小程序中的<map/>组件主要负责地理位置信息的简单展示，它没有百度地图、高德地图那样丰富的API，目前地图具备绘制图标、路线、半径等能力，在<scroll-view/>中不能使用<map/>组件，组件属性如下：

·latitude: 中心纬度。

·longitude: 中心经度。

·scale: 缩放级别，默认为16。

·markers: 标记点，用于在地图上显示标记的位置，不能自定义图标和样式，每个标记点属性如下：

·id: 标记点id, marker点击事件回调会返回此id。

·latitude: 纬度, 浮点数, 范围-90~90, 必填项。

·longitude: 经度, 浮点数, 范围-180~180, 必填项。

·title: 标注点名。

·iconPath: 显示的图标, 项目目录下的图片路径, 支持相对路径写法, 必填项。

·rotate: 旋转角度, 顺时针旋转的角度, 范围0~360, 默认为0。

·alpha: 标注的透明度, 默认1, 无透明。

·width: 标注图标宽度, 默认为图片实际宽度。

·height: 标注图标高度, 默认为图片实际高度。

·polyline: 路线, 指定一系列坐标点, 从数组第一项连线至最后一项, 数组元素属性如下:

·points: 经纬度数组, 如: [{latitude: 0, longitude: 0}], 必填项。

·color: 线的颜色, 8位十六进制表示, 后两位表示alpha值, 如: #000000AA。

·width: 线的宽度。

·dottedLine: 是否为虚线，默认为false。

·circles: 圆，数组类型，在地图上显示圆，数组元素属性如下：

·latitude: 纬度，浮点数，范围-90~90，必填项。

·longitude: 经度，浮点数，范围-180~180必填项。

·color: 描边的颜色，8位十六进制表示，后两位表示alpha值，

如：#000000AA。

·fillColor: 填充颜色，8位十六进制表示，后两位表示alpha值，

如：#000000AA。

·radius: 半径，必填项。

·strokeWidth: 描边的宽度。

·controls: 控件。

·id: 控件id，在控件点击事件回调会返回此id。

·position: 控件在地图的位置，控件相对地图位置，必填项，

position元素属性如下：

- left: 距离地图的左边界多远, 默认为0。
- top: 距离地图的上边界多远, 默认为0。
- width: 控件宽度, 默认为图片宽度。
- height: 控件高度, 默认为图片高度。
- iconPath: 显示的图标, 相对应用根目录路径的图片, 必填项。
- clickable: 是否可点击, 默认不可点击。
- include-points: 缩放视野以包含所有给定的坐标点。
- show-location: 显示带有方向的当前定位点。
- bindmarkertap: 点击标记点时触发。
- bindcontroltap: 点击控件时触发。
- bindregionchange: 视野发生变化时触发。
- bindtap: 点击地图时触发。

在下面示例中, 我们使用两个**marker**标记起始位置和结束位置, 用**polyline**绘制路线, 然后通过**controls**创建了一个按钮, 如图4-30所示。



图4-30 地图示例

代码如下:

```
<map id="map" longitude="{{longitude}}" latitude="{{latitude}}" circles="
{{circles}}" scale="14" control="{{controls}}" bindcontroltap="controltap"
markers="{{markers}}" bindmarkertap="markertap" polyline="{{polyline}}"
bindregionchange="regionchange" show-location style="width: 100%; height: 300px;">
</map>

Page({
  data: {
    latitude: 30.5491861989,
    longitude: 104.0680165911,
    scale: 5,
    // 设置标记点
    markers: [
      {iconPath: "./images/flag.png",id: 0,latitude: 30.5491861989,
longitude: 104.0680165911,width: 30,height: 30},
      {iconPath: "./images/flag.png",id: 1,latitude: 30.5468832218,
longitude: 104.0568588833,width: 30,height: 30}
    ]
  }
})
```

```

    ],
    // 设置路线
    polyline: [{
        // 按顺序设置5个点
        points: [
            {longitude: 104.0680165911, latitude: 30.5491861989},
            {longitude: 104.0687752749, latitude: 30.5493485980},
            {longitude: 104.0688698344, latitude: 30.5470634483},
            {longitude: 104.0568588833, latitude: 30.5468832218}
        ],
        color: "#0000ffDD",
        width: 2,
        dottedLine: true
    }],
    // 设置2个图标
    controls: [{
        id: 1, iconPath: './images/location.png',
        position: {left: 0, top: 250, width: 30, height: 30},
        clickable: true
    }]
},
// 地图发生变化时触发
regionchange(e) {
    // type可以区分是开始拖动还是结束拖动
    console.log(e.type)
},
// 标记点点击时触发
markertap(e) {
    // 根据标记点markerId区分
    console.log(e.markerId)
},
// 点击controls设置的图标
controltap(e) {
    // 通过markerId区分按钮
    console.log(e.controlId)
}
})

```

4.8 画布组件

`<canvas/>`主要用于绘制图形，在页面上放置一个`<canvas/>`，就相当于在页面放置了一块“画布”，可以在其中进行图形绘制。`<canvas/>`组件是一块无色透明区域，本身并没有绘制能力，它仅仅是图形容器，需要调用相关API来完成实际的绘图任务，本章主要讲解`<canvas/>`组件，相关API在后续章节中讲解。

`<canvas/>`组件默认宽度300px，高度225px，同一页面中的`canvas-id`不能重复，如果使用一个已经出现过的`canvas-id`，该`<canvas/>`组件对应的画布将被隐藏并不再正常工作，需要注意的是请勿在`<scroll-view/>`中使用`<canvas/>`。其属性如下：

- `canvas-id`: `canvas`组件的唯一标识符。
- `disable-scroll`: 当在`canvas`中移动时，禁止屏幕滚动以及下拉刷新，默认为`false`。
- `bindtouchstart`: 手指触摸动作开始。
- `bindtouchmove`: 手指触摸后移动。
- `bindtouchend`: 手指触摸动作结束。

·bindtouchcancel: 手指触摸动作被打断, 如来电提醒、弹窗。

·bindlongtaop: 手指长按500ms之后触发, 触发了长按事件后进行移动不会触发屏幕的滚动。

·binderror: 当发生错误时触发error事件, detail={errMsg: 'something wrong'}。

canvas组件本身没有太多展示的点, 这里我们简单结合API为大家初步演示canvas绘图能力, 具体绘图API参考后面章节, 示例如图4-31所示。

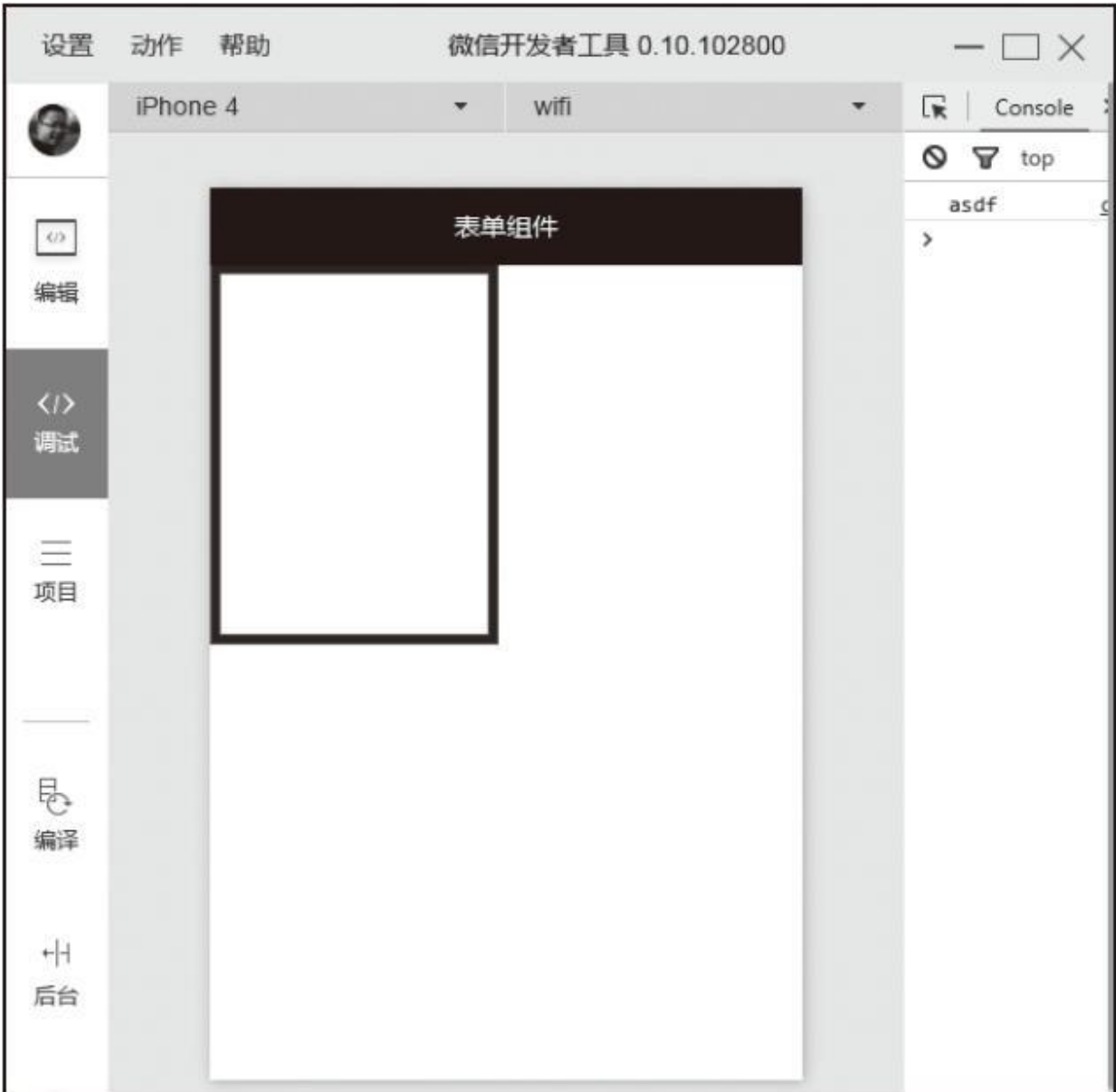


图4-31 canvas示例

代码如下:

```
<canvas canvas-id="myCanvas" style="width : 100%;height : 300px;"></canvas>

Page( {
  onReady : function() {
    // 获取绘图上下文
    var context = wx.createContext();

    context.setStrokeStyle( "#0000ff" ); // 设置线条样式
    // 设置线条宽度
```

```
context.setLineWidth( 5 ); // 设置线条宽度
context.rect(3, 2, 150, 200); // 添加一个矩阵
context.stroke(); // 对当前路径进行描边

// 绘制图像
wx.drawCanvas( {
  canvasId : 'myCanvas',
  actions : context.getActions()
} );
console.log( 'asdf' );
}
} );
```

4.9 客服会话

`<contact-button/>`用于在页面上显示一个客服会话按钮，用户点击后会进入客服会话，客服会话机制需要后台系统配合，`<contact-button/>`仅仅作为一个展示组件，目前`<contact-button/>`能力有限，仅能设置大小和图标颜色，其属性如下：

- `size`：会话按钮大小，有效值为18~27，单位px，默认值为18。
- `type`：会话按钮的样式类型，有效值为`default-dark`、`default-light`，默认值为`default-dark`。
- `session-from`：用户从按钮进入会话时，开发者将收到带上本参数的事件推送。本参数可用于区分用户进入客服会话的来源。

`<contact-button/>`使用非常简单，示例如图4-32所示。在WXML中写入标签即可唤醒会话，而客服会话中的功能则需要后台对接，具体参考5.8.5节“客服消息”。



图4-32 客服会话

代码如下：

```
<contact-button session-from="yoursession" style="margin : 100rpx;">
  </contact-button>
```

4.10 小结

本章对小程序组件进行了相应的介绍，这些组件能满足大部分开发需求，在学习过程中，大家需要掌握每个组件的特性，在项目中灵活应用，一些如媒体、地图、画布等组件的个别功能需要配合API使用，相关API将在接下来的章节中进行介绍。

第5章 API

上一章为大家介绍了小程序组件的使用，运用这些组件我们可以完成很多我们需要的UI界面，但小程序的一些功能就要依赖框架提供的API来完成。本章主要讲解小程序提供的API，包括网络、媒体、文件、数据缓存、位置、设备、界面、开放接口8大类，这些API由微信本身提供，通过逻辑层JS代码进行调用。

在学习小程序API之前，先介绍小程序API在定义和使用上的一些通用规则。一般来说，微信API按命名规则可分为两种：

- 以wx.on开头的API，如：wx.onSocketOpen、wx.onBackgroundAudioPlay（）、wx.onCompassChange（）等，均代表监听某个事件发生的API接口，这类接口一般来说参数均为一个callback回调函数，当该事件触发时，会回调callback函数。

- 其他不以wx.on开头的API，如：wx.request、wx.uploadFile、wx.chooseImage，这类接口如果没有特殊约定，通常都接受一个Object对象作为参数，在Object中可以指定success、fail、complete来接收接口调用结果，这三个回调函数在有些API中具有返回值，有些没有。

两种调用方式如下：

```
// 以wx.on开头
wx.onSocketOpen( function( res ) {
    console.log( 'WebSocket连接已打开' );
} );

// 不以wx.on开头
wx.request( {
    /* 接口调用成功 */
    success : function() {},
    /* 接口调用失败 */
    fail : function() {},
    /* 接口调用结束（调用成功、失败都会执行） */
    complete : function() {}
} );
```

5.1 网络

每个微信小程序需要事先设置一个通信域名，小程序可以跟指定的域名进行网络通信。包括不同HTTPS请求（`wx.request`）、WebSocket通信（`wx.connectSocket`）、上传文件（`wx.uploadFile`）和下载文件（`wx.downloadFile`）。设置通信域名需要登录小程序“后台->设置->开发设置，”在服务器配置中添加受信任的域。

5.1.1 发起HTTPS请求

wx.request (Object)

`request`用于发起HTTPS请求，默认超时时间和最大超时时间都是60秒。在小程序中只能使用HTTPS请求不能使用HTTP请求，一个微信小程序，同时只能有5个网络请求连接。Object参数属性如下：

·url: 后台服务器接口地址，url不能有端口必填项。

·data: 请求的参数。

·header: 设置请求的header，header中content-type默认为'application/json'，header中不能设置referer，referer格式固定为<https://servicewechat.com/{appid}/{version}/page-frame.html>，其中{appid}为小程序的appid，{version}为小程序的版本号，版本号为0表示为开发版。

·method: HTTPS请求方法类型，默认为GET，有效值有：OPTIONS、GET、HEAD、POST、PUT、DELETE、TRACE、CONNECT，method有效值必须为大写。

·**dataType**: 默认为json。如果设置了dataType为json, 则会尝试对响应的数据做一次JSON.parse。

·**success**: 收到服务器成功返回的回调函数, res={data: '开发者服务器返回的内容'}。

·**fail**: 接口调用失败的回调函数。

·**complete**: 接口调用结束的回调函数（调用成功、失败都会执行）。

data数据最终发送给服务器的数据是String类型, 如果传入的data不是String类型, 则会被转化成String。转化规则如下:

1) 对于header['content-type']为'application/json'的数据, 会对数据进行JSON序列化。

2) 对于header['content-type']为'application/x-www-form-urlencoded'的数据, 会将数据转换成query string

(encodeURIComponent (k) =encodeURIComponent (v)
&encodeURIComponent (k) =encodeURIComponent (v) ...)

示例代码如下:

```
wx.request( {  
  url : 'https://www.dmall.com',  
  header: {
```

```
        'Content-Type': 'application/json'
    },
    success : function( e ) {
        console.log( e );
    },
    complete : function() {
        console.log( '无论成功失败都会执行' );
    }
} );
```

对于request方法有以下注意事项:

- 开发者工具0.10.102800版本，header的content-type设置异常。

- 客户端的HTTPS TLS版本为1.2，但Android的部分机型还未支持TLS 1.2，所以请确保HTTPS服务器的TLS版本支持1.2及以下版本。

5.1.2 上传、下载

1.wx.uploadFile (Object)

上节中提到的request仅用于与服务端进行简单通信，如需提交带有本地资源的数据，便需要调用此接口将本地资源上传到指定服务器。调用这个uploadFile时，客户端会向服务端发起一个HTTP POST请求，header中Content-Type为multipart/form-data。上传文件最大并发限制为10个，默认超时时间和最大超时时间都是60秒，Object参数属性如下：

·url: 开发者服务器url，必填项。

·filePath: 要上传文件资源的路径，必填项。

·name: 文件对应的key，开发者在服务端通过这个key可以获取到文件的二进制内容，必填项。

·header: 设置请求的header，header中不能设置referer，referer格式固定为<https://servicewechat.com/{appid}/{version}/page-frame.html>，其中{appid}为小程序的appid，{version}为小程序的版本号，版本号为0表示为开发版。

·**formData**: HTTP请求中其他额外的请求数据。

·**success**: 收到服务器成功返回的回调函数，**success**返回参数属性如下：

·**data**: 开发者服务器返回的数据。

·**statusCode**: HTTP状态码。

·**fail**: 接口调用失败的回调函数。

·**complete**: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.chooseImage({ // 调用选中图片接口
  success : function( res ) {
    var tempFilePaths = res.tempFilePaths; // 获取临时图片地址
    wx.uploadFile( {
      url : 'http://www.myserver.com',
      filePath : tempFilePaths[0],
      name : 'myFile',
      formData : { // 其他数据
        'otherData' : 'value'
      },
      success : function( response ) {
        var data = response.data; // 服务器返回数据
      }
    } );
  }
});
```

2.wx.downloadFile (Object)

从服务端下载资源到本地。调用API后，客户端直接发起一个HTTP GET请求，把文件下载到本地并返回文件的本地临时路径，临时文件在小程序本次启动期间可以正常使用，如需持久保存，需要主动调用wx.saveFile进行保存。下载文件文件最大并发限制为10个，默认超时时间和最大超时时间都是60s，Object参数属性如下：

·url: 下载资源的URL，必填项。

·header: 设置请求的header，header中不能设置referer，referer格式固定为<https://servicewechat.com/{appid}/{version}/page-frame.html>，其中{appid}为小程序的appid，{version}为小程序的版本号，版本号为0表示为开发版。

·success: 收到服务器成功返回的回调函数，res={tempFilePath: '文件临时路径'}。文件临时路径在小程序启动期间可以正常使用，如需持久保存，需要主动调用wx.saveFile方法，这样才能保证小程序下次启动时也能访问。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.downloadFile( {  
  url : 'http://www.myserver.com', // 接口地址  
  success : function( res ) {  
    wx.getImageInfo( { // 下载图片资源后读取其信息。  
      src : res.tempFilePaths[0],  
      success : function( info ) {  
        console.log( info.width + ',' + info.height );  
      }  
    } );  
  }  
} );
```

5.1.3 WebSocket

上节中介绍的网络通信方法都是通过客户端主动向服务端发起请求，由服务端响应返回数据来达到通信的目的，这种方式有个缺点，不能实现服务端主动向客户端推送消息。采用这种短连接方式实现客户端和服务端之间的即时通信技术只能使用轮询，即在特定时间间隔（如每1秒），由客户端向服务端发起请求，然后由服务端返回最新的数据。这种方式需要客户端要不断的向服务端发出请求，在处理即时通信时，这种方式既浪费性能又占带宽。面对这种情况，在需要即时通信时，我们可以利用小程序提供的WebSocket相关API创建WebSocket，实现长连接达到即时通信的目的。长连接服务端比短连接更耗资源，在技术选项上要慎重。

1.wx.connectSocket (Object)

创建一个WebSocket连接。一个小程序同时只能有一个WebSocket连接，创建新连接时，如果当前已存在一个WebSocket连接，会自动关闭该连接，并重新创建一个新的WebSocket连接，创建连接时默认和最大超时时间都是60秒。Object参数如下：

- url: 服务器接口地址。必须是WSS协议，且域名必须是后台配置的合法域名，必填项。

·data: 请求的数据。

·header: 参考5.1.1节。

·method: 请求方式，默认为GET，有效值为：OPTIONS、GET、HEAD、POST、PUT、DELETE、TRACE、CONNECT，非必填项。

·success: 接口调用成功的回调函数。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.connectSocket( {  
  url : 'wss://www.myserver.com', // 服务端需要实现socket监听  
  data : {  
    myData : 'data'  
  },  
  header : {  
    'Content-Type' : 'application/json' // 可以不设置  
  },  
  method : 'get'  
} );
```

2.wx.onSocketOpen (callback)

监听WebSocket连接打开事件。

示例代码如下：

```
wx.connectSocket( { // 创建连接
  url : 'wss://www.myserver.com',
} );

wx.onSocketOpen( function( res ) {
  console.log( 'socket连接已打开' );
} );
```

3.wx.onSocketError (callback)

监听WebSocket错误。

示例代码如下：

```
wx.connectSocket( { // 创建连接
  url : 'wss://www.myserver.com',
} );

wx.onSocketOpen( function( res ) {
  console.log( 'socket连接已打开' );
} );

wx.onSocketError( function( res ) {
  console.log( '连接打开失败。' );
} );
```

4.wx.sendSocketMessage (callback)

通过WebSocket连接发送数据，需要先通过wx.connectSocket连接后台，并在wx.onSocketOpen回调之后才能发送。Object参数属性如下：

- data：需要发送的内容，必填项。
- success：接口调用成功的回调函数。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
var socketOpen = false;
var socketMsgQueue = [];

wx.connectSocket( { // 创建连接
  url : 'wss://www.myserver.com',
} );

wx.onSocketOpen( function( res ) { // 监听连接打开
  socketOpen = true;
  // 连接打开后先处理之前栈中的数据
  for ( var i = 0, msg; msg = socketMsgQueue[i]; ++i ) {
    sendMsg( msg );
  }
  socketMsgQueue = [];
} );

function sendMsg( msg ) {
  // 如果连接在请求中还未打开，先将数据压入信息栈中
  if ( !socketOpen ) {
    socketMsgQueue.push( msg );
    return;
  }

  // 如果连接已打开，直接发送数据
  wx.sendSocketMessage( {
    data : msg
  } );
}

sendMsg( { myName : 'weixin' } );
```

5.wx.onSocketMessage (callback)

监听WebSocket接收到服务器的消息事件。callback返回参数为data，是服务器返回的消息。

示例代码如下：

```
wx.connectSocket( {  
  url : 'wss://www.myserver.com'  
} );  
  
wx.onSocketMessage( function( res ) {  
  console.log( '收到服务器内容: ' + res.data );  
} );
```

6.wx.closeSocket ()

关闭WebSocket连接。

示例代码如下：

```
wx.closeSocket();
```

7.wx.onSocketClose (callback)

监听WebSocket是否关闭。当WebSocket成功关闭时触发。

示例代码如下：

```
wx.connectSocket( {  
  url : 'wss://www.myserver.com'  
} );  
/*  
  这里关闭socket必须在成功打开后，  
  如果由于网络或时效问题在成功打开前调用closeSocket，那么就做不到关闭WebSocket目的。  
*/  
wx.onSocketOpen( function() {  
  wx.closeSocket();  
} );  
  
wx.onSocketClose( function() {  
  console.log( '连接已关闭' );  
} );
```

5.2 媒体

5.2.1 图片

1.wx.chooseImage (Object)

从本地相册选择图片或者使用相机拍照。拍照时产生的临时路径，在小程序本次启动期间可以正常使用，如需持久保存，需要主动调用wx.saveFile，这样才能保证小程序下次启动时能访问到。Object参数属性如下：

·count: 最多可以选择的图片张数，默认为9。

·sizeType: 图片尺寸类型，有效值为original（原图）、compressed（压缩图），默认二者都有。

·sourceType: 获取图片的方式，有效值为album（从相册选图）、camera（使用相机），默认二者都有。

·success: 成功则返回图片的本地文件路径列表tempFilePaths，必填项。

·fail: 接口调用失败的回调函数。

·**complete**: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
<view>
  <image src="{{src}}"></image>
  <button bindtap="chooseImage">选择图片</button>
</view>

Page( {
  src : '',
  chooseImage : function() {
    var self = this;
    wx.chooseImage( {
      count : 1,
      sizeType : ['original', 'compressed'],
      sourceType : ['album', 'camera'],
      success : function( res ) {
        console.log( res.tempFilePaths );
      }
    } );
  }
} );
```

2.wx.previewImage (Object)

预览图片，调用后小程序会开启图片浏览界面，Object参数属性如下：

·**current**: 当前显示图片的链接，链接地址必须是urls中已存在的链接，不填则默认为urls的第一张。

·**urls**: 需要预览的图片链接列表，必填项。

·**success**: 接口调用成功的回调函数。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.previewImage( {  
  // 默认先显示第二张  
  current : 'http://www.uimaker.com/uploads/allimg/130301/1_130301090721_3.jpg',  
  urls : [  
    'http://www.uimaker.com/uploads/allimg/130301/1_130301090720_2.jpg',  
    'http://www.uimaker.com/uploads/allimg/130301/1_130301090721_3.jpg'  
  ]  
} );
```

3.wx.getImageInfo (Object)

获取图片信息，Object属性如下：

·src: 图片的路径，可以是相对路径、临时文件路径、存储文件路径，网络图片路径。

·success: 接口调用接口成功的函数，返回参数如下：Width为图片宽度，height为图片高度，单位均为px。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
chooseImage : function() {  
  wx.chooseImage( {  
    success : function( res ) {  
      wx.getImageInfo( {  
        src : res.tempFilePaths[0],  
        success : function( info ) {  
          console.log( info );  
        }  
      } );  
    }  
  } );  
}
```

5.2.2 录音

1.wx.startRecord (Object)

开始录音。当主动调用wx.stopRecord，或者录音超过1分钟时自动结束录音，返回录音文件的临时文件路径当用户离开小程序时，此接口无法调用。录音接口需要用户授权，业务代码中需要考虑用户拒绝授权的场景。Object参数属性：

·success: 录音成功后调用，返回录音文件的临时文件路径，res={tempFilePath: '录音文件的临时路径'}。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
    wx.playVoice({ // 录音完成后立即播放。
      filePath: tempFilePath
    })
  }
});
```

2.wx.stopRecord ()

主动调用停止录音。

根据上例代码，本例在10秒钟后停止录音，示例代码如下：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
    wx.playVoice({ // 录音完成后立即播放。
      filePath: tempFilePath
    });
  }
});

setTimeout( function() {
  wx.stopRecord();
}, 10000 );
```

5.2.3 音频播放控制

1.wx.playVoice (Object)

开始播放语言，同时只允许一个语音文件播放，如果前一个语音文件还没播放完，将中断前一个语音播放。Object参数属性如下：

- filePath: 需要播放的语言文件路径，必填项。
- success: 接口调用成功的回调函数。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
    wx.playVoice({ // 录音完成后立即播放。
      filePath: tempFilePath
    });
  }
});
```

2.wx.pauseVoice ()

暂停正在播放的语音。再次调用wx.playVoice播放同一个文件时，会从暂停处开始播放。如果想从头开始播放，需要先调用wx.stopVoice。

根据上例代码，我们实现5秒后自动暂停播放，示例代码如下：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
    wx.playVoice({ // 录音完成后立即播放。
      filePath: tempFilePath
    });
    setTimeout( function() {
      wx.pauseVoice();
    }, 5000 );
  }
});
```

3.wx.stopVoice ()

结束播放语音。

示例代码如下：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
    wx.playVoice({ // 录音完成后立即播放。
      filePath: tempFilePath
    });
    setTimeout( function() {
      wx.stopVoice();
    }, 5000 );
  }
});
```

5.2.4 音乐播放控制

1.wx.getBackgroundAudioPlayerState (Object)

获取后台音乐播放状态。调用本方法时可以获得<audio/>组件或wx.playBackgroundAudio () 方法播放的音乐。Object参数属性如下：

·success: 接口调用成功的回调函数，success返回参数属性如下：

·duration: 选定音频的长度（单位：s），只有在当前有音乐播放时返回。

·currentPosition: 选定音频的播放位置（单位：s），只有在当前有音乐播放时返回。

·status: 播放状态（0：暂停中，1：播放中，2：没有音乐在播放）。

·downloadPercent: 音频的下载进度（整数，如：80代表80%），只有在当前有音乐播放时返回。

·dataUrl: 歌曲数据链接，只有在当前有音乐播放时返回。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getBackgroundAudioPlayerState( {  
  success : function( res ) {  
    var statusText = {  
      2 : '没有音乐在播放',  
      1 : '播放中',  
      0 : '暂停中'  
    }  
    console.log( statusText[res.status] );  
  }  
} );
```

2.wx.playBackgroundAudio (Object)

播放音乐，同时只能有一首音乐正在播放。Object参数属性如下：

·dataUrl: 音乐链接，必填项。

·title: 音乐标题。

·coverImgUrl: 封面URL。

·success: 接口调用成功的回调函数。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.playBackgroundAudio( {
  dataUrl : './music/mymusic.mp3',
  title : '我的音乐',
  coverImgUrl : './images/poster.jpg',
  success : function() {
    console.log( '开始播放音乐了' );
  }
} );
```

3.wx.pauseBackgroundAudio ()

暂停播放音乐。

示例代码如下：

```
wx.playBackgroundAudio( {
  dataUrl : './music/mymusic.mp3',
  title : '我的音乐',
  coverImgUrl : './images/poster.jpg',
  success : function() {
    console.log( '开始播放音乐了' );
  }
} );

setTimeout( function(){
  console.log( '暂停播放' );
  wx.pauseBackgroundAudio();
}, 60000 ); // 60秒后自动暂停
```

4.wx.seekBackgroundAudio (Object)

控制音乐播放进度，利用这个API可以实现音乐的快进、快退和播放位置拖动功能，Object属性如下：

- position**: 音乐位置，单位：秒。
- success**: 接口调用成功的回调函数。
- fail**: 接口调用失败的回调函数。
- complete**: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getBackgroundAudioPlayerState( {  
  success : function( res ) {  
    var currentPosition = res.currentPosition;  
    wx.seekBackgroundAudio( {  
      position : currentPosition + 30  
    } );  
  }  
} );
```

5.wx.stopBackgroundAudio ()

停止播放音乐。

示例代码如下：

```
wx.playBackgroundAudio( {  
  dataUrl : './music/mymusic.mp3',  
  title : '我的音乐',  
  coverImgUrl : './images/poster.jpg',  
  success : function() {  
    console.log( '开始播放音乐了' );  
  }  
} );  
  
setTimeout( function(){  
  console.log( '停止播放' );  
  wx.stopBackgroundAudio();  
, 60000 ); // 60秒后自动停止
```

6.wx.onBackgroundAudioPlay (callback)

监听音乐播放。

示例代码如下：

```
wx.onBackgroundAudioPlay( function() { // 事件先绑定
  console.log( '音乐播放' );
} );

wx.playBackgroundAudio( {
  dataUrl : './music/mymusic.mp3',
  title : '我的音乐',
  coverImgUrl : './images/poster.jpg',
  success : function() {
    console.log( '开始播放音乐了' );
  }
} );
```

7.wx.onBackgroundAudioPause (callback)

监听音乐暂停。

示例代码如下：

```
wx.onBackgroundAudioPause( function() {
  console.log( '音乐暂停' );
} );

wx.playBackgroundAudio( {
  dataUrl : './music/mymusic.mp3',
  title : '我的音乐',
  coverImgUrl : './images/poster.jpg',
  success : function() {
    console.log( '开始播放音乐了' );
  }
} );
```

8.wx.onBackgroundAudioStop (callback)

监听音乐停止。

示例代码如下：

```
wx.onBackgroundAudioStop( function() {  
    console.log( '音乐停止' );  
} );  
  
wx.playBackgroundAudio( {  
    dataUrl : './music/mymusic.mp3',  
    title : '我的音乐',  
    coverImgUrl : './images/poster.jpg',  
    success : function() {  
        console.log( '开始播放音乐了' );  
    }  
} );
```

5.2.5 音频组件控制

`wx.createAudioContext (audioId)` 创建并返回audio上下文 `audioContext` 对象，`audioContext` 通过 `audioId` 和一个 `<audio/>` 组件绑定，通过它可以操作对应的 `<audio/>` 组件。 `audioContext` 对象方法如下：

- `setSrc`： 设置音频地址。
- `play`： 播放。
- `pause`： 暂停。
- `seek`： 跳转到指定位置， 单位为s。

利用播放相关API配合 `<audio/>` 我们可以创建自定义样式播放器，示例如图5-1所示。



图5-1 自定义样式播放器

示例代码如下：

```
<view class="custom-audio">
  <audio id="myAudio" style="display:none;"></audio>
  <image class="poster" mode="widthFix"
src="http://www.yoka.com/dna/pics/ba1c1117/42/d3551ec1c17cadcd1.jpg"></image>
  <view class="actions" bindtap="action">
    <view data-type="play">播放</view>
    <view data-type="pause">暂停</view>
    <view data-type="seek">跳到30秒</view>
    <view data-type="reset">回到开头</view>
  </view>
</view>
```

```

    </view>
</view>

.custom-audio { width : 500rpx; border : solid 1px #ccc; border-bottom : 0;
margin : 60rpx auto; }
.custom-audio .poster { width : 80%; border-radius: 50%; margin: 10%; }
.custom-audio .actions view { height : 80rpx; line-height: 80rpx; text-align:
center; border: solid 1px #ccc; border-width : 1px 0px; background:-webkit-
gradient(linear, 0% 0%, 0% 100%,from(#fff), to(#eee)); margin-top: 10rpx; }

Page( {
  onReady : function( e ) {
    this.audioContext = wx.createAudioContext( 'myAudio' );
    this.audioContext.setSrc(
'http://ws.stream.qqmusic.qq.com/M5000001VfvsJ21xFqb.mp3?
guid=ffffffff82def4af4b12b3cd9337d5e7&uin=346897220&vkey=6292F51E1E384E06DCBDC9AB
7C49FD713D632D313AC4858BACB8DDD29067D3C601481D36E62053BF8DFEAF74C0A5CCFADD6471160
CAF3E6A&fromtag=46' );
  },
  action : function( e ) {
    var type = e.target.dataset.type,
        audioContext = this.audioContext;

    switch ( type ) {
      // 播放
      case 'play' :
        audioContext.play();
        break;
      // 暂停
      case 'pause' :
        audioContext.pause();
        break;
      // 跳到30秒处
      case 'seek' :
        audioContext.seek( 30 );
        break;
      // 从头播放
      case 'reset' :
        audioContext.seek( 0 );
        break;
    }
  }
} );

```

5.2.6 视频

`wx.chooseVideo` (Object) 拍摄视频或从手机相册中选取视频，返回视频的临时文件路径。Object参数属性如下：

·`sourceType`: 获取视频方式，`album`从相册选中视频，`camera`使用相机拍摄，默认值为：['album', 'camera']。

·`maxDuration`: 拍摄视频最长拍摄时间，单位秒。最长支持60秒。

·`camera`: 拍摄适配的方式用前置或后置摄像头，默认为前后都有：['album', 'camera']。

·`success`: 接口调用成功，返回视频文件的临时文件路径，返回参数属性如下：

·`tempFilePath`: 选定视频的临时文件路径。

·`duration`: 选定视频的时间长度。

·`size`: 选定视频的数据量大小。

·`height`: 选定视频的高。

·`width`: 选定视频的高。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.chooseVideo( {  
  sourceType : [ 'album', 'camera' ],  
  camera : [ 'front', 'back' ],  
  success : function( res ) {  
    console.log( res.tempFilePath );  
  }  
} );
```

5.2.7 视频组件控制

`wx.createVideoContext (videoId)` 同 `wx.createAudioContext (audioId)` 一样，`wx.createVideoContext (videoId)` 用于创建并返回 video 上下文 `videoContext` 对象，`videoContext` 对象方法如下：

·`play`：播放。

·`pause`：暂停。

·`seek`：跳转到指定位置，单位 s。

·`sendDanmu`：发送弹幕，`danmu` 包含两个属性 `text`，`color`。

示例代码如下：

```
<video src="./video/myvideo.mp4" id="myVideo"></video>

<button bindtap="play">播放</button>
<button bindtap="pause">暂停</button>
<button bindtap="restart">回到开头</button>
<button bindtap="sendDanmu">发送弹幕</button>

Page( {
  onReady : function() {
    this.videoContext = wx.createVideoContext( 'myVideo' );
  },
  play : function() {
    this.videoContext.play();
  },
  pause : function() {
    this.videoContext.pause();
  },
  restart : function() {
    this.videoContext.seek( 0 );
  },
  sendDanmu : function() {
    this.videoContext.sendDanmu( {
      text : '弹幕',
      color : 'red'
    } );
  }
})
```

```
        text : '弹幕文案',  
        color : '#ff0000'  
    } );  
}  
} );
```

5.3 文件

从网络下载、拍照、录音、录视频时，文件都是存在临时文件中，需要永久保存这些文件就需要我们主动调用API进行保存，这时小程序会将文件保存到系统指定目录，关于这些文件操作都需要调用相关API。

1.wx.saveFile (Object)

保存文件到本地，本地文件存储大小限制为10MB。Object参数属性如下：

- tempFilePath: 需要保存文件的临时路径，必填项。
- success: 返回文件的保存路径，res={savedFilePath: '文件的保存路径'}。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```

wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath // 临时文件路径
    wx.saveFile({ // 保存临时文件
      tempFilePath: tempFilePath,
      success: function(res) {
        var savedFilePath = res.savedFilePath; // 永久地址路径
        console.log( '录音文件已保存到: ' + savedFilePath );
      }
    })
  }
});

```

2.wx.getSavedFileList (Object)

获取本地已保存文件列表，Object参数属性如下：

·success: 接口调用成功的回调函数，success返回参数属性如下：

·errMsg: 接口调用结果。

·fileList: 文件列表，fileList项目属性包括：

·filePath: 文件的本地路径。

·createTime: 文件保存的时间，从1970/01/01 08: 00: 00到当前时间的秒数。

·size: 文件的大小，单位B。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getSavedFileList( {  
  success : function( res ) {  
    for ( var i = 0, file; file = res.fileList[i]; ++i ) {  
      console.log( '第' + i + '个文件路径: ' + file.filePath );  
    }  
  }  
} );
```

3.wx.getSavedFileInfo (Object)

获取本地文件的文件信息，Object属性如下：

·filePath: 文件路径，必填项。

·success: 接口调用成功的回调函数，success返回参数属性如下：

·errMsg: 接口调用结果。

·size: 文件大小，单位B。

·createTime: 文件保存时的时间戳，从1970/01/01 08: 00: 00到当前时间的秒数。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getSavedFileList( {
  success : function( res ) {
    for ( var i = 0, file; file = res.fileList[i]; ++i ) {
      wx.getSavedFileInfo( {
        filePath : file.filePath,
        success : function( res ) {
          console.log( '文件大小为: ' + res.size );
        }
      } );
    }
  }
} );
```

4.wx.removeSavedFile (Object)

删除本地存储的文件，Object参数属性如下：

- filePath: 需要删除的文件路径，必填项。
- success: 接口调用成功的回调函数。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getSavedFileList( {
  success : function( res ) {
    // 删除所有文件
    for ( var i = 0, file; file = res.fileList[i]; ++i ) {
      wx.removeSavedFile( {
        filePath : file.filePath
      } );
    }
  }
} );
```

5.wx.openDocument (Object)

在新页面中打开文档，支持格式有：doc、docx、xls、xlsx、ppt、pptx、pdf、Object参数属性如下：

- filePath: 文件路径，可通过wx.downloadFile获得，必填项。
- success: 接口调用成功的回调函数。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.downloadFile( {  
  url : 'http://www.myserver.com/my.docx',  
  success : function( res ) {  
    var filePath = res.filePath;  
    // 下载文档后在新页面中预览  
    wx.openDocument( {  
      filePath : filePath  
    } );  
  }  
} );
```

5.4 数据缓存

每个小程序都可以有自己的本地缓存，**local Storage**是永久存储的，本地缓存最大为**10MB**，数据操作**API**分为同步和异步两种。

5.4.1 保存数据

1.wx.setStorage (Object)

将数据存在本地缓存指定的key中，会覆盖掉原来该key对应的内容，这是一个异步接口。Object参数属性如下：

- key: 本地缓存中的指定的key，必填项。
- data: 需要存储的内容，必选项，必填项。
- success: 接口调用成功的回调函数。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.setStorage( {  
  key : 'myKey',  
  value : 'myValue'  
} );
```

2.wx.setStorageSync (KEY, DATA)

将数据存在本地缓存指定的key中，会覆盖掉原来该key对应的内容，这是一个同步接口。参数说明：

·key：本地缓存中指定的key。

·data：需要存储的内容。

示例代码如下：

```
wx.setStorageSync( 'myKey', 'myValue' );
```

5.4.2 获取数据

1.wx.getStorage (Object)

从本地缓存中异步获取指定key对应的内容，Object参数属性如下：

- key: 本地缓存中的指定的key，必填项。
- success: 接口调用的回调函数，res={data: key对应的内容}，必填项。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getStorage( {  
  key : 'myKey',  
  success : function( res ){  
    console.log( res.data ); // 打印出myKey对应的内容  
  }  
} );
```

2.wx.getStorageSync (KEY)

从本地缓存中同步获取指定key的对应内容，参数说明：

key：本地缓存中的指定的key，必填项。

示例代码如下：

```
var value = wx.getStorageSync( 'myKey' );  
console.log( value );
```

5.4.3 获取本地数据信息

1.wx.getStorageInfo (Object)

异步获取当前storage的相关信息，Object参数属性如下：

- success: 接口调用的回调函数，返回参数属性如下：
- keys: 当前storage中所有的key。
- currentSize: 当前占用的空间大小，单位kb。
- limitSize: 限制的空间大小，单位kb。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getStorageInfo( {  
  success : function( res ) {  
    var p;  
    for ( p in res.keys ) {  
      console.log( p + ':' + wx.getStorageSync( p ) );  
    }  
  }  
} );
```

2.wx.getStorageSync ()

同步获取当前storage的相关信息。

示例代码如下：

```
var p,
    info = wx.getStorageSync();
for ( p in info.keys ) {
    console.log( p + ':' + wx.getStorageSync( p ) );
}
```

5.4.4 删除数据

1.wx.removeStorage (Object)

根据key值异步删除本地数据，Object参数属性如下：

- key: 本地缓存中指定的key，必填项。
- success: 接口调用的回调函数，必填项。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.removeStorage( {  
  key : 'myKey',  
  success : function( res ) {  
    console.log( res.data );  
  }  
} );
```

2.wx.removeStorageSync (Key)

根据key值同步删除本地数据。

示例代码如下：

```
wx.removeStorageSync( 'myKey' );
```

5.4.5 清空数据

1.wx.clearStorage ()

清理本地数据缓存。

示例代码如下：

```
wx.clearStorage();  
//不阻塞下面代码  
doSomething();
```

2.wx.clearStorageSync ()

同步清理本地数据缓存。

示例代码如下：

```
wx.clearStorageSync();  
//阻塞下面代码  
doSomething();
```

5.5 位置

5.5.1 获取位置

在国际上，坐标体系有多套标准，小程序支持WGS84标准和GCJ02标准，WGS84是地球坐标系，国际上通用的坐标系。设备一般包含的GPS芯片或者北斗芯片所获取的经纬度为WGS84地理坐标系。GCJ02坐标系为火星坐标系，是由中国国家测绘局制订的地理信息系统的坐标系统，它是由WGS84坐标系经加密后的坐标系，它是在小程序中，查看位置需要使用GCJ02标准坐标。

`wx.getLocation` (Object) 用于获取当前的地理位置、速度，需要用户开启定位功能，当用户离开小程序后，此接口无法调用；当用户点击“显示在聊天顶部”时，此接口可继续调用，Object参数属性如下：

- type: 默认为wgs84返回gps坐标，gcj02返回可用于wx.openLocation的坐标。

- success: 接口调用成功的回调函数，必填项，返回参数属性如下：

- latitude: 纬度，浮点数，范围为-90～90，负数表示南纬。
- longitude: 经度，浮点数，范围-180～180，负数表示西经。
- speed: 速度，浮点数，单位m/s。
- accuracy: 位置的精确度。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getLocation( {  
  type : 'wgs84',  
  success : function( res ) {  
    console.log( res );  
  }  
} );
```

5.5.2 选择位置

`wx.chooseLocation` (Object) 用于打开地图选择位置，用户选中后返回选中信息，Object参数属性如下：

·`success`: 接口调用成功的回调函数，必填项，`success`返回参数属性如下：

·`name`: 位置名称。

·`address`: 详细地址。

·`latitude`: 纬度，浮点数，范围为-90~90，负数表示南纬。

·`longitude`: 经度，浮点数，范围为-180~180，负数表示西经。

·`cancel`: 用户取消时调用。

·`fail`: 接口调用失败的回调函数。

·`complete`: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.chooseLocation( {  
  success : function( res ) {  
    console.log( res.address );  
  }  
} );
```

5.5.3 查看位置

`wx.openLocation` (Object) 用于在微信内置地图查看位置，Object 参数属性如下：

- `latitude`: 纬度，范围为-90~90，负数表示南纬，必填项。
- `longitude`: 经度，范围为-180~180，负数表示西经，必填项。
- `scale`: 缩放比例，范围1~28，默认为28。
- `name`: 位置名。
- `address`: 地址的详细说明。
- `success`: 接口调用成功的回调函数。
- `fail`: 接口调用失败的回调函数。
- `complete`: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getLocation( {  
  type : 'gcj02', // 返回可用于wx.openLocation的经纬度  
  success : function( res ) {  
    wx.open( { // 显示当前地址
```

```
        latitude : res.latitude,  
        longitude : res.longitude  
    } );  
}  
} );
```

5.5.4 地图组件控制

`wx.createMapContext` (Object) 用于创建并返回map上下文mapContext对象，map-Context通过mapId跟一个<map/>组件绑定，通过它可以操作对应的<map/>组件，map-Context对象方法如下：

·`getCenterLocation`：获取当前地图中心的经纬度，返回的是gcj02坐标系，可以用于`wx.openLocation`。getCenterLocation参数属性如下：

·`success`：接口调用成功的回调函数，`res={longitude: "经度", latitude: "纬度"}`。

·`fail`：接口调用失败的回调函数。

·`complete`：接口调用结束的回调函数（调用成功、失败都会执行）。

·`moveToLocation`：将地图中心移动到当前定位点，需要配合map组件的show-location使用。

示例代码如下：

```
<map id="myMap" show-location/>
<button data-type="getCenterLocation" bindtap="action">获取位置</button>
<button data-type="location" bindtap="action">移动位置</button>

Page( {
```

```
onReady : function( e ) {
    this.mapContext = wx.createMapContext( 'myMap' );
},
action : function( e ) {
    var type = e.target.dataset.type,
        mapContext = this.mapContext;

    switch ( type ) {
        // 获取当前地图中心纬度
        case 'getCenterLocation':
            mapContext.getCenterLocation( {
                success : function( res ) {
                    console.log( res.longitude + ',' + res.latitude );
                }
            } );
            // 定位到当前位置
        case 'location':
            mapContext.moveToLocation();
    }
}
} );
```

5.6 设备

5.6.1 系统信息

1.wx.getSystemInfo (Object)

异步获取系统信息。Object参数属性为

·success: 接口调用成功回调函数，返回系统相关信息，必填项，返回参数属性如下：

·model: 手机型号。

·pixelRatio: 设备像素比。

·windowWidth: 窗口宽度。

·windowHeight: 窗口高度。

·language: 微信设置的语言。

·version: 微信版本号。

·system: 操作系统版本。

·platform: 客户端平台。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getSystemInfo( {  
  success : function( info ) {  
    console.log( info );  
  }  
} );
```

2.wx.getSystemInfoSync ()

同步获取系统信息。

示例代码如下：

```
var info = wx.getSystemInfoSync();  
console.log( info );
```

5.6.2 网络状态

`wx.getNetworkType` (Object) 用于获取网络类型，Object参数属性如下：

- `success`: 接口调用成功回调函数，返回网络类型`networkType`，必填项。

- `fail`: 接口调用失败的回调函数。

- `complete`: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getNetworkType( {  
  success : function( res ) {  
    console.log( res.networkType );  
  }  
} );
```

5.6.3 重力感应

`wx.onAccelerometerChange` (callback) 用于监听重力感应数据，频率：5次/秒，callback返回参数属性如下：

·x: X轴重力感应，值为当前轴上重力加速度/重力加速度
(9.8) 。

·y: Y轴重力感应，值参考x轴。

·z: Z轴重力感应，值参考x轴。

示例代码如下：

```
<view>{{x}},{{y}},{{z}}</view>

Page( {
  data : {x : 0,y : 0,z : 0},
  onReady : function() {
    var self = this;
    wx.onAccelerometerChange(function(res) {
      self.setData( {x : res.x,y : res.y,z : res.z} );
    })
  }
} );
```

5.6.4 罗盘

`wx.onCompassChange`（callback）用于监听罗盘数据，频率：5次/秒，调用罗盘需要开启定位功能，callback参数的属性有direction：当前面向的方向度数，正北方为0，范围为0~360，-1代表没有开启定位功能。

示例代码如下：

```
wx.onCompassChange( function( res ) {  
  console.log( res.direction );  
} );
```

5.6.5 拨打电话

`wx.makePhoneCall` (Object) 用于调用手机拨打电话功能，Object 参数的属性如下：

- `phoneNumber`: 需要拨打的电话号码，必填项。
- `success`: 接口调用成功回调函数。
- `fail`: 接口调用失败的回调函数。
- `complete`: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.makePhoneCall( {  
  phoneNumber : '1345678910'  
} );
```

5.6.6 扫码

`scanCode` (Object) 调起客户端扫码界面，扫码成功后返回对应结果。Object参数如下：

- `success`: 接口调用成功的回调函数，返回参数如下：
- `result`: 扫码的内容。
- `scanType`: 所扫码的类型。
- `charSet`: 所扫码的字符集。
- `path`: 当所扫的码为当前小程序的合法二维码时，会返回此字段，内容为二维码携带的path。
- `fail`: 调用接口失败的回调函数。
- `complete`: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.scanCode( {  
  success : function( res ) {  
    // 打印扫码内容  
    console.log( res.result );  
  }  
})
```

```
}  
} );
```

5.7 界面

5.7.1 交互反馈

1.wx.showToast (Object)

显示消息提示框，Object参数的属性如下：

- title: 提示内容，必填项。
- icon: 图标，只支持”success”、”loading”。
- duration: 提示的延迟时间，到了指定时间自动关闭，单位毫秒，默认为1500，最大为10000。
- mask: 是否显示透明蒙层，防止触摸穿透，默认为false。
- success: 接口调用成功回调函数。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.showToast( {  
  title : '操作成功',  
  icon : 'success'  
} );
```

2.wx.hideToast ()

隐藏消息提示框。

示例代码如下：

```
wx.showToast( {  
  title : '请稍等...',  
  icon : 'loading',  
  duration : 10000  
} );  
  
setTimeout( function() { // 或请求返回时关闭toast  
  wx.hideToast();  
}, 2000 );
```

3.wx.showModal (Object)

显示模态弹窗，Object参数的属性如下：

- title: 提示的标题，必填项。
- content: 提示的内容，必填项。
- showCancel: 是否显示取消按钮，默认为true。
- cancelText: 取消按钮的文字，默认为“取消”，最多为4个字符。

- cancelColor: 取消按钮的文字颜色，默认为“#000000”。
- confirmText: 确定按钮的文字，默认为“确定”，最多4个字符。
- confirmColor: 确定按钮的文字颜色，默认为“#3CC51F”
- success: 接口调用成功回调函数，返回结果res.confirm为true时，表示用户点击确定按钮。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

在Android微信6.6.30版中，wx.showModal返回的confirm一直为true。

示例代码如下：

```
wx.showModal( {  
  title : '标题',  
  content : '内容',  
  success : function( res ) {  
    if ( !res.confirm ) {  
      return; // 用户点击了取消，不作任何处理  
    }  
    console.log( '用户点击了确定' ); // 可以继续流程  
  }  
} );
```

4.wx.showActionSheet (Object)

显示操作菜单，Object参数的属性如下：

·itemList: 按钮的文字数组，数组长度最大为6个，必填项。

·itemColor: 按钮的文字颜色，默认为"#000000"。

·success: 接口调用成功回调函数，返回参数如下：

·cancel: 用户是否取消选择。

·tapIndex: 用户点击的按钮，从上到下的顺序，从0开始。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.showActionSheet( {  
  itemList : ['缺货', '配送时间选错', '不想买了', '其他'], // 订单用户取消原因  
  success : function( res ) {  
    if ( !res.cancel ) {  
      console.log( res.tapIndex );  
    }  
  }  
} );
```

5.7.2 设置导航条

1.wx.setNavigationBarTitle (Object)

动态设置当前页面的标题，Object参数属性如下：

- title: 页面标题，必填项。
- success: 接口调用成功的回调函数。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.setNavigationBarTitle( {  
  title : '新页面'  
} );
```

2.wx.showNavigationBarLoading ()

在当前页面显示导航条加载动画。

示例代码如下：

```
Page( {  
  onShow : function() {  
    wx.showNavigationBarLoading();  
  }  
} );
```

3.wx.hideNavigationBarLoading ()

隐藏导航条加载动画。

示例代码如下：

```
Page( {  
  onLaunch : function() {  
    wx.hideNavigationBarLoading();  
  }  
} );
```

5.7.3 导航

1.wx.navigateTo (Object)

保留当前页面，跳转到应用内的某个页面，使用wx.navigateBack可以返回到原页面，小程序中页面路径最多5层。Object参数的属性如下：

·url: 需要跳转的应用内页面的路径，路径后可以带参数，必填项。参数与路径之间使用“?”分隔，参数键与参数值用“=”相连，不同参数用“&”分隔；如'path? key=value&key2=value2'。

·success: 接口调用成功的回调函数。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.navigateTo( {  
  url : '../newpage?key1=value1'  
} );  
  
Page( {  
  onLoad : function( option ) {  
    console.log( option.query );  
  }  
});
```

```
}  
} );
```

2.wx.redirectTo (Object)

关闭当前页面，跳转到应用内的某个页面，Object参数属性如下：

·url: 需要跳转的应用内页面的路径，路径后可以带参数，必填项。参数与路径之间使用? 分隔，参数键与参数值用=相连，不同参数用&分隔；如'path? key=value&key2=value2'。

·success: 接口调用成功的回调函数。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.redirectTo( {  
  url : '../newpage?key1=value1'  
} );
```

3.wx.switchTab (Object)

跳转到tabBar页面，并关闭其他所有非tabBar页面。Object参数的属性如下：

·url: 需要跳转的tabBar页面的路径（需在app.json的tabBar字段定义的页面），路径后不能带参数，必填项。

·success: 接口调用成功的回调函数。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
// app.json配置tabBar
{
  "tabBar" : {
    "list" : [{
      "pagePath" : "home",
      "text" : "首页"
    }, {
      "pagePath" : "other",
      "text" : "其他页面"
    }]
  }
}
wx.switchTab( {
  url : '/home' // 必须是已注册tab
} );
```

4.wx.navigateBack (Object)

关闭当前页面，返回上一页面或多级页面。可通过 `getCurrentPages`（）获取当前的页面栈，以决定返回几层。Object参数的属性为delta：返回的页面数，如果delta大于现有的页面数，则返回到首页，默认值为1。

示例代码如下：

```
// 向前返回3级
wx.navigateBack( { delta : 3 } );
```

5.7.4 动画

`wx.createAnimation` (Object) 用于创建一个动画实例`animation`。可以调用动画实例的方法来描述动画，最后通过动画实例的`export`方法导出动画数据，传递给组件`animation`属性，每次调用`exports`方法会清掉之前的动画操作。Object参数的属性如下：

·`duration`: 动画持续时间，单位`ms`，默认值`400`。

·`timingFunction`: 定义动画的效果，默认值`"linear"`，有效值为`"linear"`、`"ease"`、`"ease-in"`、`"ease-in-out"`、`"ease-out"`、`"step-start"`、`"step-end"`。

·`delay`: 动画延迟时间，单位`ms`，默认值`0`。

·`transformOrigin`: 设置`transform-origin`，默认为`"50%50%0"`。

示例代码如下：

```
var animation = wx.createAnimation( {  
  transformOrigin : '0 0', // 已左上角为中心点  
  duration : 20000,  
  timingFunction : 'ease-in'  
} );
```

1.动画描述

创建实例后我们需要调用实例动画方法来描述动画，这些方法调用后会返回自身，支持链式调用的写法，动画描述方法按类型可分为样式、旋转、缩放、偏移、倾斜、矩阵变形，下面分别介绍。

(1) 样式

·opacity (value) : 设置透明度，参数说明：

·value: 透明度，参数范围0~1。

·backgroundColor (color) : 设置背景颜色，参数说明：

·color: 颜色值。

·width (length) : 设置宽度，参数说明：

·length: 长度值，如果传入Number则默认使用px，可传入其他自定义单位的长度值。

·height (length) : 设置高度，参数说明：

·length: 长度值，如果传入Number则默认使用px，可传入其他自定义单位的长度值。

·top (length) : 设置top属性，参数说明：

·length: 长度值, 如果传入Number则默认使用px, 可传入其他自定义单位的长度值。

·left (length): 设置left属性, 参数说明:

·length: 长度值, 如果传入Number则默认使用px, 可传入其他自定义单位的长度值。

·bottom (length): 设置bottom属性, 参数说明:

·length: 长度值, 如果传入Number则默认使用px, 可传入其他自定义单位的长度值。

·right (length): 设置right属性, 参数说明:

·length: 长度值, 如果传入Number则默认使用px, 可传入其他自定义单位的长度值。

(2) 旋转

·rotate (deg): 从原点顺时针旋转一个deg角度, 参数说明:

·deg: 旋转角度, 范围-180~180。

·rotateX (deg): 在X轴旋转一个deg角度, 参数说明:

·deg: 旋转角度, 范围-180~180。

·rotateY (deg) : 在Y轴旋转一个deg角度, 参数说明:

·deg: 旋转角度, 范围-180~180°。

·rotateZ (deg) : 在Z轴旋转一个deg角度, 参数说明:

·deg: 旋转角度, 范围-180~180°。

·rotate3d (x, y, z, deg) : 旋转的简写方法, 参数说明:

·x: 在X轴的旋转角度, 范围-180~180°。

·y: 在y轴的旋转角度, 范围-180~180°。

·z: 在z轴的旋转角度, 范围-180~180°。

·deg: 从原点顺时针旋转的角度, 范围-180~180°。

(3) 缩放

·scale (sx, sy) : 同时设置在X轴、Y轴的缩放, 参数说明:

·sx: 一个参数时, 表示在X轴、Y轴同时缩放sx倍数; 两个参数时表示在X轴缩放sx倍数。

·sy: Y轴缩放的sy倍数, 可以不填写。

·scaleX (sx) : 设置X轴缩放, 参数说明:

·sx: X轴缩放的sx倍数。

·scaleY (sy) : 设置Y轴缩放倍数, 参数说明:

·sy: Y轴缩放的sy倍数。

·scaleZ (sz) : 设置Z轴缩放倍数, 参数说明:

·sz: Z轴缩放的sz倍数。

·scale3d (sx, sy, sz) : 同时控制X、Y、Z轴缩放:

·sx: X轴缩放的sx倍数。

·sy: Y轴缩放的sy倍数。

·sz: Z轴缩放的sy倍数。

(4) 偏移

·translate (tx, ty) : 表示在X、Y轴偏移量, 参数说明:

·tx: 一个参数时, 表示在X轴偏移tx, 单位px; 两个参数时, 表示在X轴偏移tx。

·ty: 在Y轴偏移ty, 单位px。

·translateX (tx) : 表示在X轴偏移量, 参数说明:

·tx: 在X轴偏移tx, 单位px。

·translateY (ty) : 表示在Y轴偏移量, 参数说明:

·ty: 在Y轴偏移ty, 单位px。

·translateZ (tz) : 表示在Z轴偏移量, 参数说明:

·tz: 在Z轴偏移tz, 单位px。

·translate3d (tx, ty, tz) : 表示在X、Y、Z轴偏移量, 参数说明:

·tx: 在X轴偏移tx, 单位px。

·ty: 在Y轴偏移ty, 单位px。

·tz: 在Z轴偏移tz, 单位px。

(5) 倾斜

·skew (ax, ay) : 表示在X、Y轴倾斜, 参数说明:

·ax: 该方法有一个参数ax时, Y轴坐标不变, X轴坐标延顺时针倾斜ax度; 有两个参数 (ax, ay) 时, 表示在X轴倾斜ax度。

·ay: , 在Y轴倾斜ay度。

·**skewX (ax)**：表示在X轴倾斜，参数说明：

·**ax**：X轴倾斜ax度。

·**skewY (ay)**：表示在Y轴倾斜，参数说明：

·**ay**：，在Y轴倾斜ay度。

(6) 矩阵变形

·**matrix (a, b, c, d, tx, ty)**：定义矩阵变形，基于X轴和Y轴坐标重新定位元素位置，同CSS3中**transform-function matrix**。

·**matrix3d ()**：定义矩阵变形，基于X轴、Y轴、Z轴重新定位元素位置，同CSS3中**transform-function matrix3d**。

矩阵变形相对复杂，小程序矩阵变形同CSS3中的矩阵变形方法一样，大家可以参考网络资料。

2.动画队列

调用动画操作方法后要调用**step ()**来表示一组动画完成，可以在一组动画中调用任意多个动画方法，一组动画中的所有动画会同时开始，一组动画完成后才会进行下一组动画。**step**可以传入一个跟**wx.createAnimation ()**一样的配置参数，用于指定当前组动画的配

置。在iOS/Android微6.3.30版中，通过step（）分隔动画时，只有第一步动画能生效。

示例代码如下：

```
<view style="width:100%;height:200px;">
  <icon animation="{{animData}}" style="position:absolute;left:100px;"
  type="success" size="40"/>
</view>
<button bindtap="rotateAndMove">旋转并移动</button>
<button bindtap="rotateThenMove">旋转后移动</button>

Page({
  data: {
    animData: {}
  },
  // 旋转并移动
  rotateAndMove : function() {
    var anim = wx.createAnimation( {
      duration : 1000,
      timingFunction : 'ease'
    } );
    // 同时执行2个动画
    anim.translateY( '100px' ).rotate( '720' ).step();
    this.setData( {
      animData : anim.export()
    } );
  },

  // 旋转后再移动
  rotateThenMove : function() {
    var anim = wx.createAnimation( {
      timingFunction : 'ease'
    } );
    // 分步执行3个动画，通过step切割
    anim.rotate( '720' ).step( { duration : 500 } )
    anim.translateY( '100px' ).step( { duration : 500 } );
    anim.translateX( '100px' ).step( { duration : 500 } );
    this.setData( {
      animData : anim.export()
    } );
  }
});
```

5.7.5 绘图

上章提到过<canvas/>组件，在<canvas/>组件中画图必须通过本小结介绍的API实现，<canvas/>中存在一个二维坐标体系，左上角的坐标为（0，0），所有绘图API都会基于这个坐标体系进行绘制。绘图过程大致可分为3步：

- 1) 创建执行上下文。
- 2) 通过执行上下文进行绘制描述。
- 3) 调用绘图API，将绘制描述绘制到到<canvas/>。

示例代码如下，绘图演示如图5-2所示。

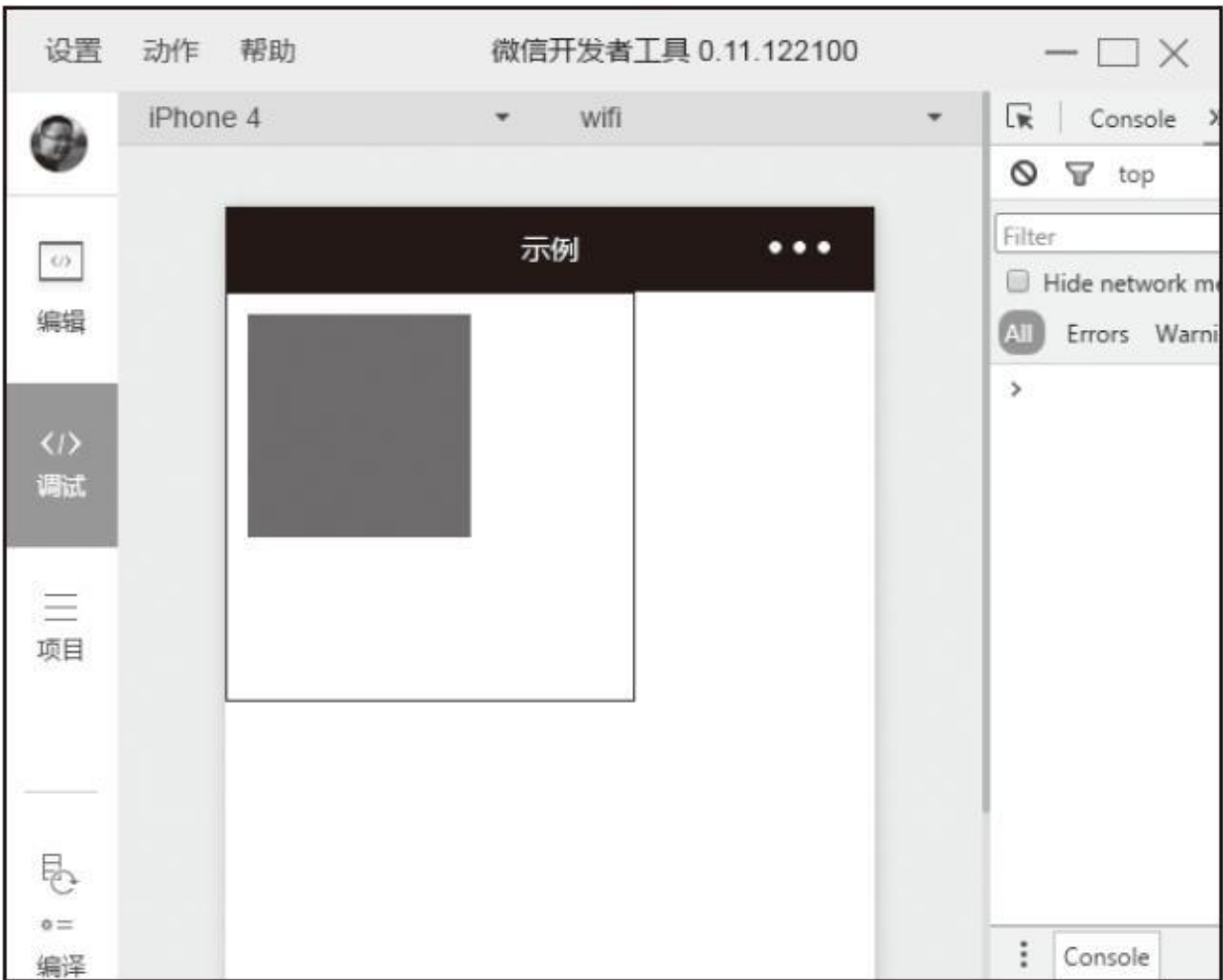


图5-2 绘图演示

```
<canvas canvas-id="myCanvas" style="width:200px; height:200px; border:solid 1px"/>

Page( {
  onReady : function() {
    // 第1步，创建执行上下文
    var canvasContext = wx.createCanvasContext( 'myCanvas' );
    // 第2步，设置绘制描述
    canvasContext.setFillStyle( 'red' );
    canvasContext.fillRect( 10, 10, 110, 110 );
    // 第3步，绘图
    canvasContext.draw();
  }
} );
```

1.基础API

(1) wx.createCanvasContext (canvasId)

在绘图时我们需要调用wx.createCanvasContext () 创建一个绘图上下文context对象，一个canvasContext通过canvasId与<canvas/>一一绑定， canvasContext仅作用于对应的<canvas/>，通过调用绘图上下文相关方法实现绘图功能。

示例代码如下：

```
var context = wx.createCanvasContext( 'myCanvas' );
```

(2) wx.canvasToTempFilePath (Object)

将当前画布内容导出生成图片，并返回文件路径。Object参数的属性如下：

- canvasId: 画布标识，指定导出哪个画布，传入定义在<canvas/>的canvas-id。

- success: 接口调用成功的回调函数，返回参数属性有tempFilePath: 导出图片的文件路径。

- fail: 接口调用失败的回调函数。

- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.canvasToTempFilePath( {  
  canvasId : 'myCanvas',  
  success : function( res ) {  
    // 打印出图片路径  
    console.log( res.tempFilePath );  
  }  
} );
```

2.canvasContext对象方法

<canvas/>所有绘图行为都需要调用绘图上下文相关方法实现，这些方法整体可分为10类：样式、渐变、线条样式、矩阵、路径、变形、文字、图片、混合、其他。

(1) 样式

样式设置会全局作用于全局，如果没有覆盖，后续绘图的颜色将沿用之前的设置，样式设置相关方法如下：

·**setFillStyle (color)**：设置填充颜色，没有设置默认为black。参数为color：设置为填充样式的颜色，参数可为颜色字符串，取值范围：'rgb (255, 0, 0) '、'rgba (255, 0, 0, 0.6) '、'#ff0000'格式的颜色字符串；也可为渐变颜色对象。

·**setStrokeStyle (color)**：设置描边颜色。参数为color，同上。

·setShadow (offsetX, offsetY, blur, color) : 设置阴影, 参数说明如下:

·offsetX: 阴影相对于形状在水平方向的偏移。

·offsetY: 阴影相对于形状在竖直方向的偏移。

·blur: 阴影的模糊级别, 数值越大越模糊。

·color: 阴影的颜色, 取值范围: 'rgb (255, 0, 0)' 或 'rgba (255, 0, 0, 0.6)' 或 '#ff0000' 格式的颜色字符串。

示例代码如下, 效果见图5-3:

```
<canvas canvas-id="myCanvas" style="width:100%; height:400px;"/>
Page( {
  onReady : function() {
    var canvasContext = wx.createCanvasContext( 'myCanvas' );
    // 填充方形
    canvasContext.setFillStyle( 'red' );
    canvasContext.fillRect( 10, 10, 50, 50 );

    // 绘制方框
    canvasContext.setStrokeStyle( '#0000ff' );
    canvasContext.strokeRect( 70, 10, 50, 50 );

    // 绘制带阴影方形
    canvasContext.setShadow( 10, 10, 50, 'rgb( 0, 255, 0 )' );
    // 这里填充的方形颜色受到第一句代码影响, 也为红色
    canvasContext.fillRect( 130, 10, 50, 50 );

    canvasContext.draw();
  }
} );
```



图5-3 样式方法示例

(2) 渐变

渐变相关方法用于创建渐变对象，渐变对象可作为参数传入绘图相关方法，创建渐变的坐标是基于`canvas`的坐标，而不是被填充图形的坐标，在使用渐变填充图像时一定要注意坐标的转化，创建渐变相关方法如下：

·createLinearGradient (x0, y0, x1, y1) : 创建一个线性渐变, 参数如下:

·x0: 起点的x坐标。

·y0: 起点的y坐标。

·x1: 终点的x坐标。

·y1: 终点的y坐标。

·createCircularGradient (x, y, r) : 创建一个圆形的渐变, 起点在圆心, 终点在圆环, 参数如下:

·x: 圆心的x坐标。

·y: 圆心的y坐标。

·r: 圆的半径。

·addColorStop (stop, color) : 创建一个颜色渐变点, 参数说明:

·stop: 表示渐变点在起点和终点中的位置, 取值范围0~1。

·color: 渐变点的颜色, 取值范围: 'rgb (255, 0, 0)' 或 'rgba (255, 0, 0, 0.6)' 或 '#ff0000' 格式的颜色字符串。

示例代码如下，效果见图5-4:

```
<canvas canvas-id="myCanvas" style="width:100%; height:400px;"/>

Page( {
  onReady : function() {
    var canvasContext = wx.createCanvasContext( 'myCanvas' ),
        linearGradient, circularGradient, colorStop;

    // 需要根据图形对渐变坐标进行换算
    linearGradient = canvasContext.createLinearGradient(0, 0, 100, 0);
    // 创建渐变时，至少创建2个渐变点，开始与结束
    linearGradient.addColorStop(0, 'black');
    linearGradient.addColorStop(1, 'red');
    // 填充方形
    canvasContext.setFillStyle( linearGradient );
    canvasContext.fillRect( 10, 10, 100, 100 );

    // 需要根据图形对渐变坐标进行换算
    circularGradient = canvasContext.createCircularGradient( 170,60,50 );
    // 可以创建多个渐变点
    circularGradient.addColorStop(0, 'red');
    circularGradient.addColorStop(0.16, 'orange');
    circularGradient.addColorStop(0.33, 'yellow');
    circularGradient.addColorStop(0.5, 'green');
    circularGradient.addColorStop(0.66, 'cyan');
    circularGradient.addColorStop(0.83, 'blue');
    circularGradient.addColorStop(1, 'purple');
    // 填充方形
    canvasContext.setFillStyle( circularGradient );
    canvasContext.fillRect( 120, 10, 100, 100 );

    canvasContext.draw();
  }
} );
```

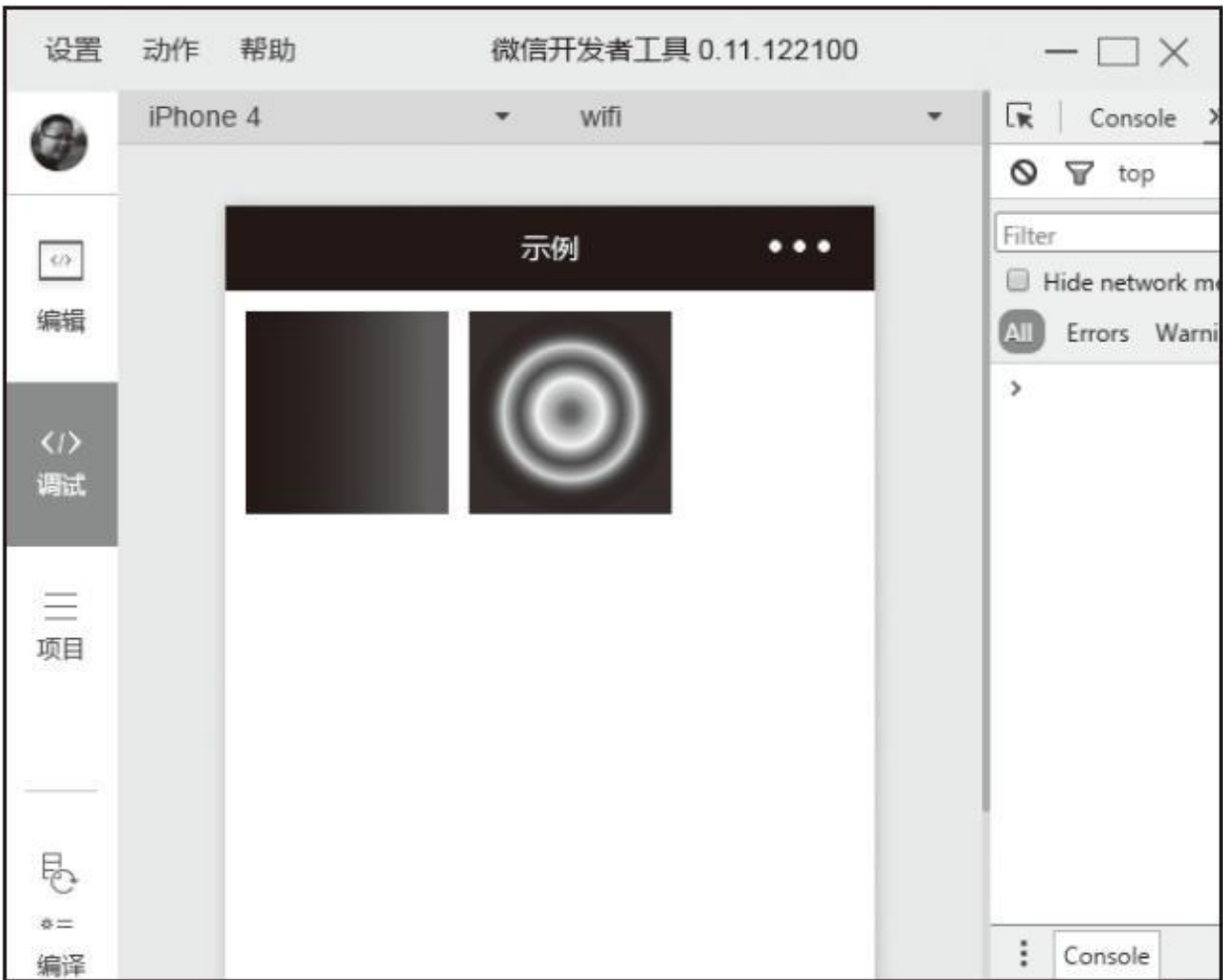


图5-4 渐变方法示例

(3) 线条样式

和样式设置一样，线条样式设置会全局作用于全局，如果没有覆盖，后续绘图的设置将沿用之前的设置，线条样式方法如下：

·`setLineWidth (lineWidth)`：设置线条的宽度，默认线条宽度为1px。参数包括lineWidth：线条的宽度，单位为px。

·**setLineCap (lineCap)**：设置线条结束端点样式，默认结束端点样式为square。参数说明包括lineCap：线条的结束端点样式，取值有：

·square：端点样式为正方形，不会截取线条宽度。

·butt：端点样式为正方形，会截取线条宽度

·round：端点样式为圆形，不会截取线条宽度。

·**setLineJoin (lineJoin)**：设置线条的交点样式，默认交叉点样式为miter。参数包括lineJoin：线条的结束交叉点样式，取值有：

·bevel：平角。

·round：圆角。

·miter：尖角。

·**setMiterLimit ()**：设置最大斜接长度，斜接长度指的是在两条线交汇处内角和外角之间的距离。当setLineJoin为miter时才有效。超过最大倾斜长度的，连接处将以lineJoin为bevel来显示，参数包括miterLimit：最大斜接长度。

示例代码如下，效果见图5-5：

```
<canvas canvas-id="myCanvas" style="width:100%; height:700px;"/>
var _fn;
```

```

Page( {
  onReady : function() {
    var canvasContext = wx.createCanvasContext( 'myCanvas' );

    // 设置线宽度
    canvasContext.setLineWidth( 10 );

    // 端点样式为正方形，不会截取线条宽度
    canvasContext.setLineCap( 'square' );
    _fn.drawLine( canvasContext, [10, 20], [150, 20] );
    // 端点样式为正方形，但会截取线条宽度
    canvasContext.setLineCap( 'butt' );
    _fn.drawLine( canvasContext, [10, 40], [150, 40] );
    // 端点样式为圆形，不会截取线条宽度
    canvasContext.setLineCap( 'round' );
    _fn.drawLine( canvasContext, [10, 60], [150, 60] );

    // 恢复为默认线条边界
    canvasContext.setLineCap( 'square' );

    // 交叉处为平角
    canvasContext.setLineJoin('bevel');
    _fn.drawAngle( canvasContext, [10, 80] );
    // 交叉处为圆角
    canvasContext.setLineJoin('round');
    _fn.drawAngle( canvasContext, [50, 80] );
    // 交叉处为尖角
    canvasContext.setLineJoin('miter');
    _fn.drawAngle( canvasContext, [90, 80] );

    // 设置交叉处为尖角
    canvasContext.setLineJoin('miter');

    canvasContext.setMiterLimit(1);
    _fn.drawAngle( canvasContext, [10, 140] );

    canvasContext.setMiterLimit(2);
    _fn.drawAngle( canvasContext, [50, 140] );

    canvasContext.setMiterLimit(3);
    _fn.drawAngle( canvasContext, [90, 140] );

    canvasContext.draw();
  }
});

_fn = {
  drawLine : function( canvasContext, start, end ) {
    canvasContext.beginPath()
    canvasContext.moveTo(start[0], start[1])
    canvasContext.lineTo(end[0], end[1])
    canvasContext.stroke();
  },

  drawAngle : function( canvasContext, beginPoint ) {
    canvasContext.beginPath()
    canvasContext.moveTo(beginPoint[0], beginPoint[1]);
    canvasContext.lineTo(beginPoint[0] + 40, beginPoint[1] + 20)
    canvasContext.lineTo(beginPoint[0], beginPoint[1] + 40)
    canvasContext.stroke()
  }
}

```



图5-5 线条样式示例

(4) 矩形

矩形相关方法如下：

·`rect (x, y, width, height)`：创建一个矩形，创建后需要调用`fill` () 或`stroke` () 方法将矩形真正画到`<canvas/>`中。参数说明如下：

·`x`：矩形路径左上角的`x`坐标。

·y: 矩形路径左上角的y坐标。

·width: 矩形路径的宽度。

·height: 矩形路径的高度。

·fillRect (x, y, width, height) : 填充一个矩形, 填充时需要调用setFillStyle () 设置颜色, 如果没设置默认是黑色。参数说明如下:

·x: 矩形路径左上角的x坐标。

·y: 矩形路径左上角的y坐标。

·width: 矩形路径的宽度。

·height: 矩形路径的宽度。

·strokeRect (x, y, width, height) : 画一个非填充的矩形, 绘制前需要调用setFillStroke () 设置线条颜色, 如没有设置默认是黑色。参数说明如下:

·x: 矩形路径左上角的x坐标。

·y: 矩形路径左上角的y坐标。

·width: 矩形路径的宽度。

·height: 矩形路径的宽度。

·clearRect (x, y, width, height) : 清除画布上在该矩形区域内的内容, 清除后会直接显示出<canvas/>背景色。参数说明如下:

·x: 矩形路径左上角的x坐标。

·y: 矩形路径左上角的y坐标。

·width: 矩形路径的宽度。

·height: 矩形路径的宽度。

示例代码如下, 效果见图5-6:

```
<canvas canvas-id="myCanvas" style="width:100%; height:700px;"/>

Page( {
  onReady : function() {
    var canvasContext = wx.createCanvasContext( 'myCanvas' );
    // 绘制线框矩形
    canvasContext.rect( 10, 10, 30, 30 );
    canvasContext.stroke();
    canvasContext.draw();
    // 绘制填充矩形
    canvasContext.rect( 50, 10, 30, 30 );
    canvasContext.fill();
    canvasContext.draw( true ); // 传入参数true, 表示接着上次继续绘制

    // 绘制填充矩形
    canvasContext.fillRect( 10, 50, 30, 30 );
    canvasContext.draw( true );
    // 绘制线框矩形
    canvasContext.strokeRect( 50, 50, 30, 30 );
    canvasContext.draw( true );

    // 清除矩形区域
    canvasContext.clearRect( 25, 25, 40, 40 );
    canvasContext.draw( true );
  }
} );
```



图5-6 矩形方法示例

(5) 路径

路径绘制并不会在画布上绘制形状，而是创建一个线条路径，创建的路径可被`stroke()`绘制线条，也可被`fill()`填充区域，绘制路径开始时需要调用`beginPath()`方法，结束时需要调用`closePath()`方法，上文`rect()`方法可认为是绘制矩形路径的一个快捷方法，通过路径方法组合，我们可以绘制任何形状图形。路径方法如下：

·**beginPath ()** : 开始创建一个路径, 需要调用**fill**或者**stroke**才会使用路径进行填充或描边。在最开始的时候相当于调用了一次**beginPath ()**。同一个路径内的多次**setFillStyle ()**、**setStrokeStyle ()**、**setLineWidth ()** 等设置, 以最后一次设置为准。

·**closePath ()** : 关闭一个路径。关闭路径会连接起点和终点。如果关闭路径后没有调用**fill ()** 或者**stroke ()** 并开启了新的路径, 那之前的路径将不会被渲染。

·**fill ()** : 对当前路径中的内容进行填充。默认的填充色为黑色。如果当前路径没有闭合, **fill ()** 方法会将起点和终点进行连接, 然后填充。**fill ()** 填充的路径是从**beginPath ()** 开始计算, 但是不会将**fillRect ()** 包含进去。

·**stroke ()** : 画出当前路径的边框。默认颜色为黑色。**stroke ()** 描绘的路径是从**beginPath ()** 开始计算, 但是不会将**strokeRect ()** 包含进去。

·**moveTo (x, y)** : 把路径移动到画布中的指定点, 但不创建线条。参数说明如下:

·**x**: 目标位置的**x**坐标。

·**y**: 目标位置的**y**坐标。

·**lineTo** (x, y) : 添加一个新点，然后在画布中创建从该点到最后指定点的线条。参数包括x、y，同上。

·**arc** (x, y, r, sAngle, eAngle, counterclockwise) : 添加一个弧形路径到当前路径，顺时针绘制，创建一个圆可以用**arc** () 方法指定其弧度为0，终止弧度为2*Math.PI。参数说明如下：

·**x**: 圆的x坐标。

·**y**: 圆的y坐标。

·**r**: 圆的半径。

·**sAngle**: 起始弧度，单位弧度（在3点钟方向）。

·**eAngle**: 终止弧度。

·**counterclockwise**: 可选。指定弧度的方向是逆时针还是顺时针。默认是false，即顺时针。

·**quadraticCurveTo** (cp_x, cp_y, x, y) : 创建二次方贝塞尔曲线，曲线的起始点为路径中前一个点。参数说明如下：

·**cp_x**: 贝塞尔控制点的x坐标。

·**cp_y**: 贝塞尔控制点的y坐标。

·x: 结束点的x坐标。

·y: 结束点的y坐标。

·bezierCurveTo (cp1x, cp1y, cp2x, cp2y, x, y) : 创建三次方贝塞尔曲线，曲线的起始点为路径中前一个点。参数说明如下：

·cp1x: 第一个贝塞尔控制点的x坐标。

·cp1y: 第一个贝塞尔控制点的y坐标。

·cp2x: 第二个贝塞尔控制点的x坐标。

·cp2y: 第二个贝塞尔控制点的y坐标。

·x: 结束点的x坐标。

·y: 结束点的y坐标。

示例代码如下，效果见图5-7:

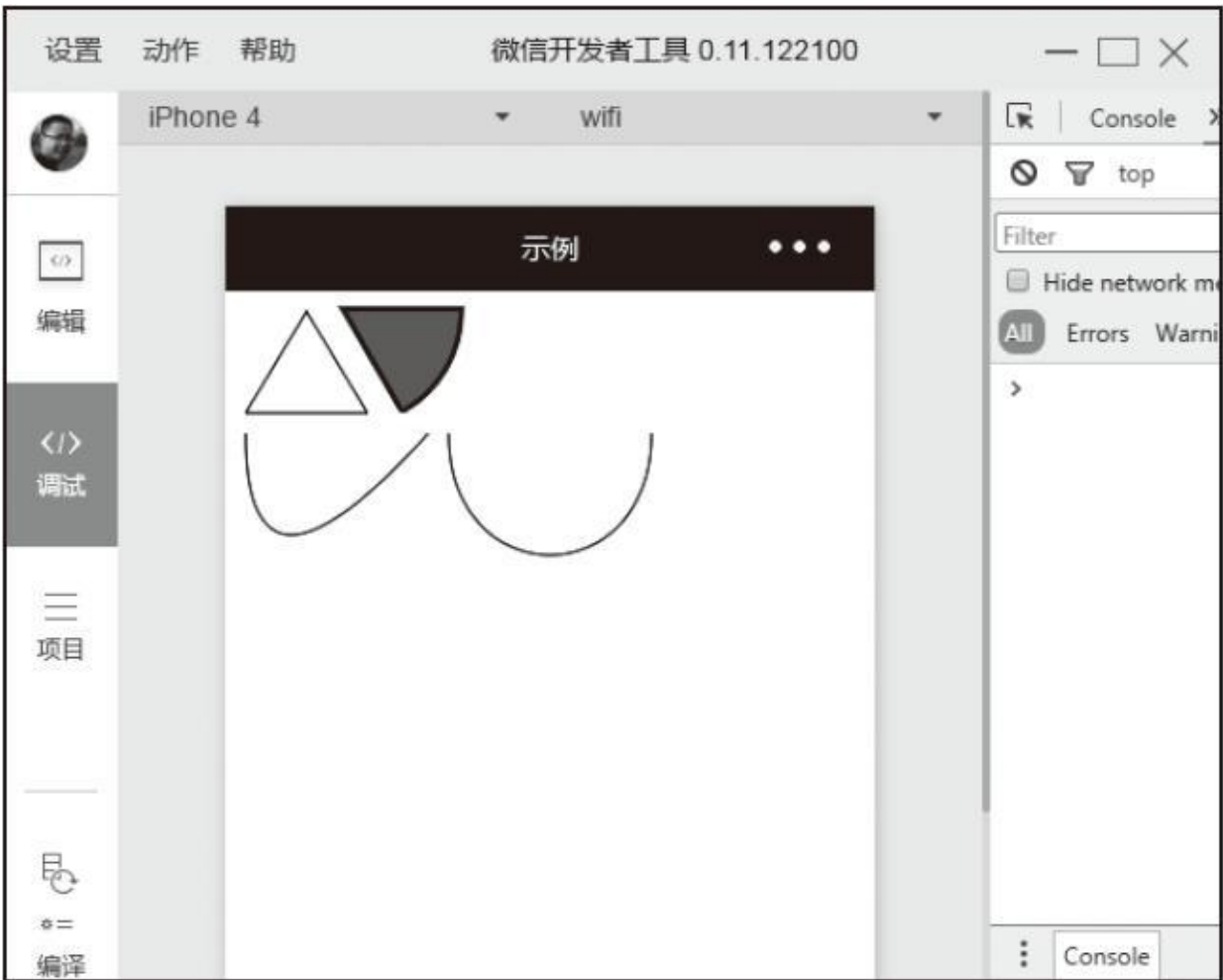


图5-7 路径方法示例

```
<canvas style="width: 100%; height: 700px;" canvas-id="myCanvas"></canvas>

var _fn;

Page( {
  onReady : function() {
    var context = wx.createCanvasContext( 'myCanvas' );

    // 绘制三角形线条
    _fn.drawTrianglePath( context, [10, 60] );
    context.stroke();
    context.draw();
    // 绘制扇形
    _fn.drawSectorPath( context, [60, 10], 55, 0, 60 );
    context.setStrokeStyle( 'black' );
    context.setLineWidth( 5 );
    context.setFillStyle( 'gray' );
    context.stroke();
    context.fill();
    context.draw( true );
    // 绘制二次贝塞尔曲线
    context.setLineWidth( 1 );
```

```

        _fn.drawQuadraticCurve( context, [10, 70], [10, 170], [100, 70] );
        context.stroke();
        context.draw( true );
        // 绘制三次贝塞尔曲线
        _fn.drawBezierCurve( context, [110, 70], [110, 150], [210, 150], [210, 70] );
        context.stroke();
        context.draw( true );
    }
} );

_fn = {
    // 绘制三角形
    drawTrianglePath : function( canvasContext, beginPos ) {
        canvasContext.moveTo( beginPos[0], beginPos[1] );
        // 开始路径
        canvasContext.beginPath();
        // 绘制3条边
        canvasContext.lineTo( beginPos[0] + 30, beginPos[1] - 50 );
        canvasContext.lineTo( beginPos[0] + 60, beginPos[1] );
        canvasContext.lineTo( beginPos[0], beginPos[1] );
        // 关闭路径
        canvasContext.closePath();
    },
    // 绘制扇形
    drawSectorPath : function( canvasContext, beginPos, radius, startAngle, endAngle ) {
        canvasContext.moveTo( beginPos[0], beginPos[1] );
        // 开始路径
        canvasContext.beginPath();
        canvasContext.lineTo( beginPos[0] + radius, beginPos[1] ); // 绘制边线
        canvasContext.arc( beginPos[0], beginPos[1], radius, startAngle*Math.PI / 180,
endAngle*Math.PI/180 ); // 绘制弧线
        canvasContext.lineTo( beginPos[0], beginPos[1] ); // 绘制边线
        // 开始路径
        canvasContext.closePath();
    },
    // 绘制二次贝塞尔曲线路径
    drawQuadraticCurve : function( canvasContext, startPoint, controlPoint, endPoint ) {
        canvasContext.beginPath();
        canvasContext.moveTo( startPoint[0], startPoint[1] );
        canvasContext.quadraticCurveTo( controlPoint[0], controlPoint[1], endPoint[0],
endPoint[1] );
        // 这里不需要关闭路径，否则路径将自动首尾相连
    },
    // 绘制三次贝塞尔曲线路径
    drawBezierCurve : function( canvasContext, startPoint, controlPoint1,
controlPoint2, endPoint ) {
        canvasContext.beginPath();
        canvasContext.moveTo( startPoint[0], startPoint[1] );
        canvasContext.bezierCurveTo( controlPoint1[0], controlPoint1[1],
controlPoint2[0], controlPoint2[1], endPoint[0], endPoint[1] );
        // 这里不需要关闭路径，否则路径将自动首尾相连
    }
}
}

```

(6) 变形

变形相关方法会整体影响之后绘图方法的坐标体系，每个方法之前可以相互迭代，可以认为变形是对整体坐标体系的变形。变形相关方法如下：

·**scale** (**scaleWidth**, **scaleHeight**)：在调用**scale**方法后，之后创建的路径其横纵坐标会被缩放。多次调用**scale**，倍数会相乘。缩放后不仅影响图形坐标，同时影响线条宽度，可认为是对画布整体进行了缩放。参数说明如下：

·**scaleWidth**：横坐标缩放的倍数（1=100%，0.5=50%，2=200%）。

·**scaleHeight**：纵坐标轴缩放的倍数（1=100%，0.5=50%，2=200%）。

·**rotate** (**rotate**)：以原点为中心，原点可以用**translate**方法修改。顺时针旋转当前坐标轴。多次调用**rotate**，旋转的角度会叠加。参数包括**rotate**：旋转角度，以弧度计（ $\text{degrees} \times \text{Math.PI} / 180$ ，**degrees**范围为0～360）。

·**translate** (**x**, **y**)：对当前坐标系的原点（0，0）进行变换，默认的坐标系原点为页面左上角。参数说明如下：

·**x**：水平坐标平移量。

·y: 竖直坐标平移量。

本节示例中多种变形效果可相互叠加，示例代码如下，效果见图5-8:

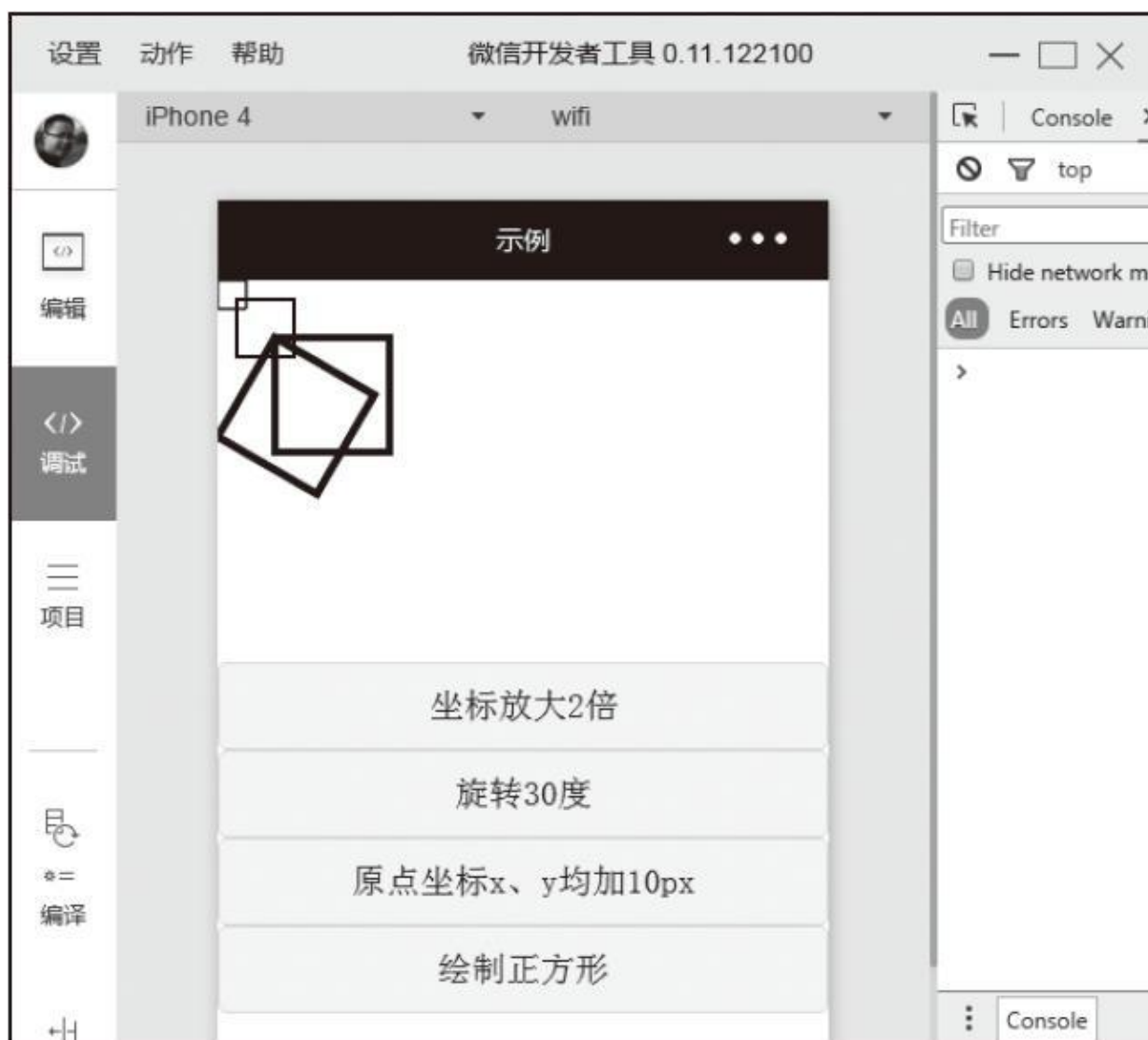


图5-8 变形方法示例

```
<canvas style="width: 100%; height: 200px;" canvas-id="myCanvas"></canvas>
<button bindtap="scale">坐标放大2倍</button>
<button bindtap="rotate">旋转30度</button>
<button bindtap="translate">原点坐标x、y均加10px</button>
<button bindtap="drawReact">绘制正方形</button>
```

```

Page( {
  canvasContext : null,
  onReady : function() {
    this.canvasContext = wx.createCanvasContext( 'myCanvas' );
  },
  translate : function() {
    this.canvasContext.translate( 10, 10 ); // 横纵坐标均移动10px
  },
  rotate : function() {
    this.canvasContext.rotate( 30 * Math.PI / 180 ); // 旋转30度
  },
  scale : function() {
    this.canvasContext.scale( 2, 2 ); // x,y均放大2倍，放大后，线条也相应放大
  },
  drawReact : function() {
    var context = this.canvasContext;
    context.restore(); // 恢复绘制条件,并弹栈。
    context.rect( 0, 0, 15, 15 );
    context.stroke();
    context.draw( true ); // 基于上次图形继续绘制
  }
} );

```

(7) 文字

文字相关方法能将文字输出到画布中，字体颜色能通过`setFillStyle`

() 方法修改，其方法包括`fillText (text, x, y)`：在画布上绘制被填充的文本。参数说明如下：

- `text`：在画布上输出的文本。

- `x`：绘制文本的左上角x坐标位置。

- `y`：绘制文本的左上角y坐标位置。

- `setFontSize (fontSize)`：设置字体的字号。参数包括`fontSize`：字体的字号。

示例代码如下，效果见图5-9:

```
<canvas style="width: 100%; height: 200px;" canvas-id="myCanvas"></canvas>

Page( {
  onReady : function() {
    var context = wx.createCanvasContext( 'myCanvas' );

    context.fillText('中文', 10, 20)
    // 设置字体大小
    context.setFontSize( 24 );
    // 改变字体颜色
    context.setFillStyle( 'red' );
    context.fillText('English', 100, 20)

    context.draw();
  }
} );
```

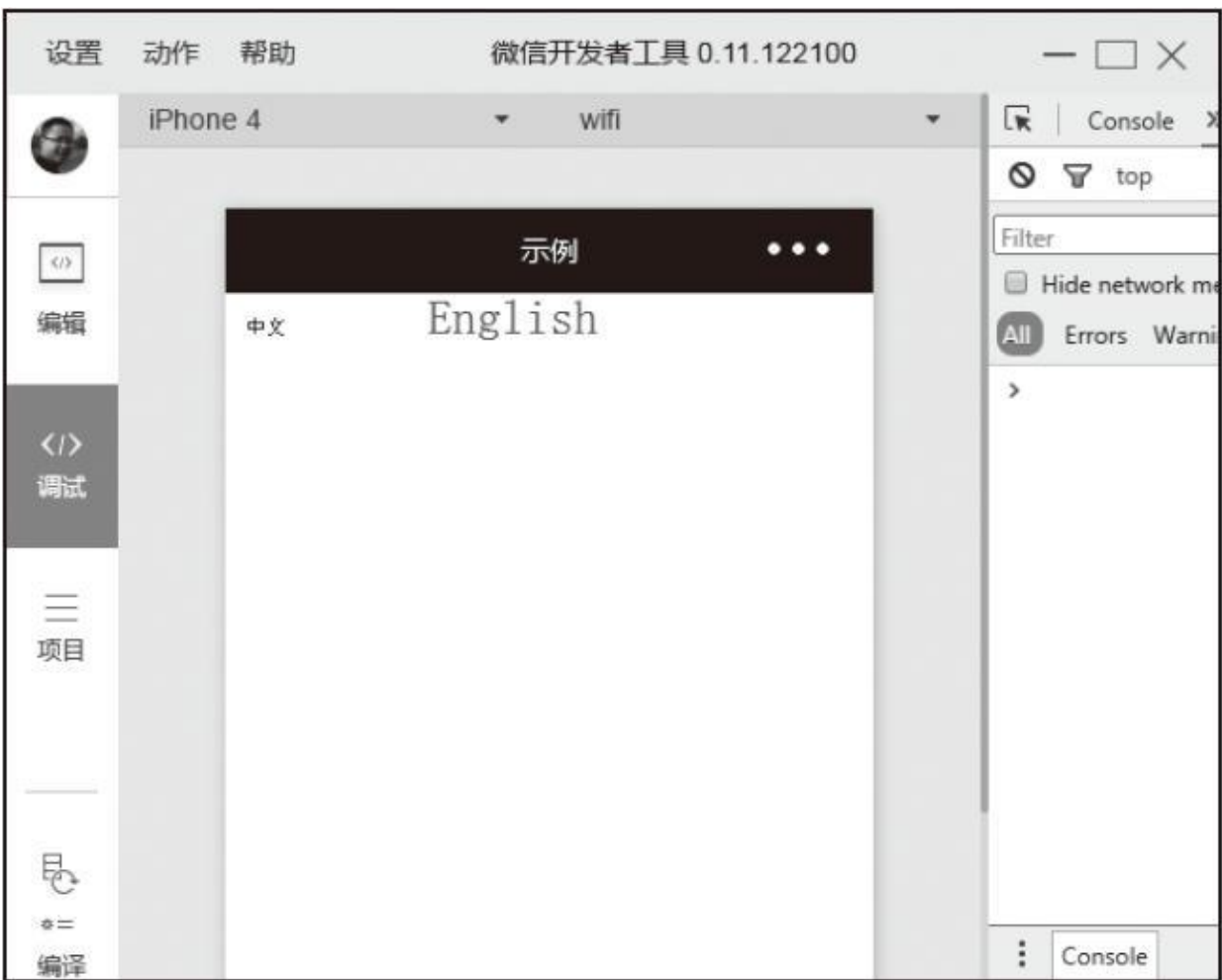


图5-9 字体方法示例

(8) 图片

图片方法只有一个drawimage (imageResource, x, y, width, height)：用于在画布上绘制图片。方法如下：

·imageResource: 所要绘制的图片资源，可以是网络资源，也可以是本地资源相对路径。

·x: 图像左上角的x坐标。

·y: 图像左上角的y坐标。

·width: 图像宽度。

·height: 图像高度。

示例代码如下，效果见图5-10:

```
<canvas style="width: 100%; height: 200px;" canvas-id="myCanvas"></canvas>

Page( {
  onReady : function() {
    var context = wx.createCanvasContext( 'myCanvas' );
    context.drawImage( 'http://www.qq1234.org/uploads/allimg/140526/
3_140526112044_2.jpg', 1, 1, 100, 131 );
    context.draw();
  }
} );
```



图5-10 图片方法

(9) 混合

混合方法只有一个`setGlobalAlpha (alpha)`：设置全局画笔透明度。参数包括`alpha`：透明度，0表示完全透明，1表示完全不透明。

示例代码如下，效果见图5-11：

```
<canvas style="width: 100%; height: 300px;" canvas-id="myCanvas"></canvas>

Page( {
  onReady : function() {
```

```

var context = wx.createCanvasContext( 'myCanvas' );

context.setFillStyle( 'blue' );
context.fillRect(10, 10, 100, 100);
context.setGlobalAlpha( 0.5 );
context.fillRect(50, 50, 100, 100);
// 对图片也生效 context.drawImage( 'http://www.qq1234.org/uploads/allimg/
140526/3_140526112044_2.jpg', 80, 80, 100, 131 );
context.draw();
}
} );

```

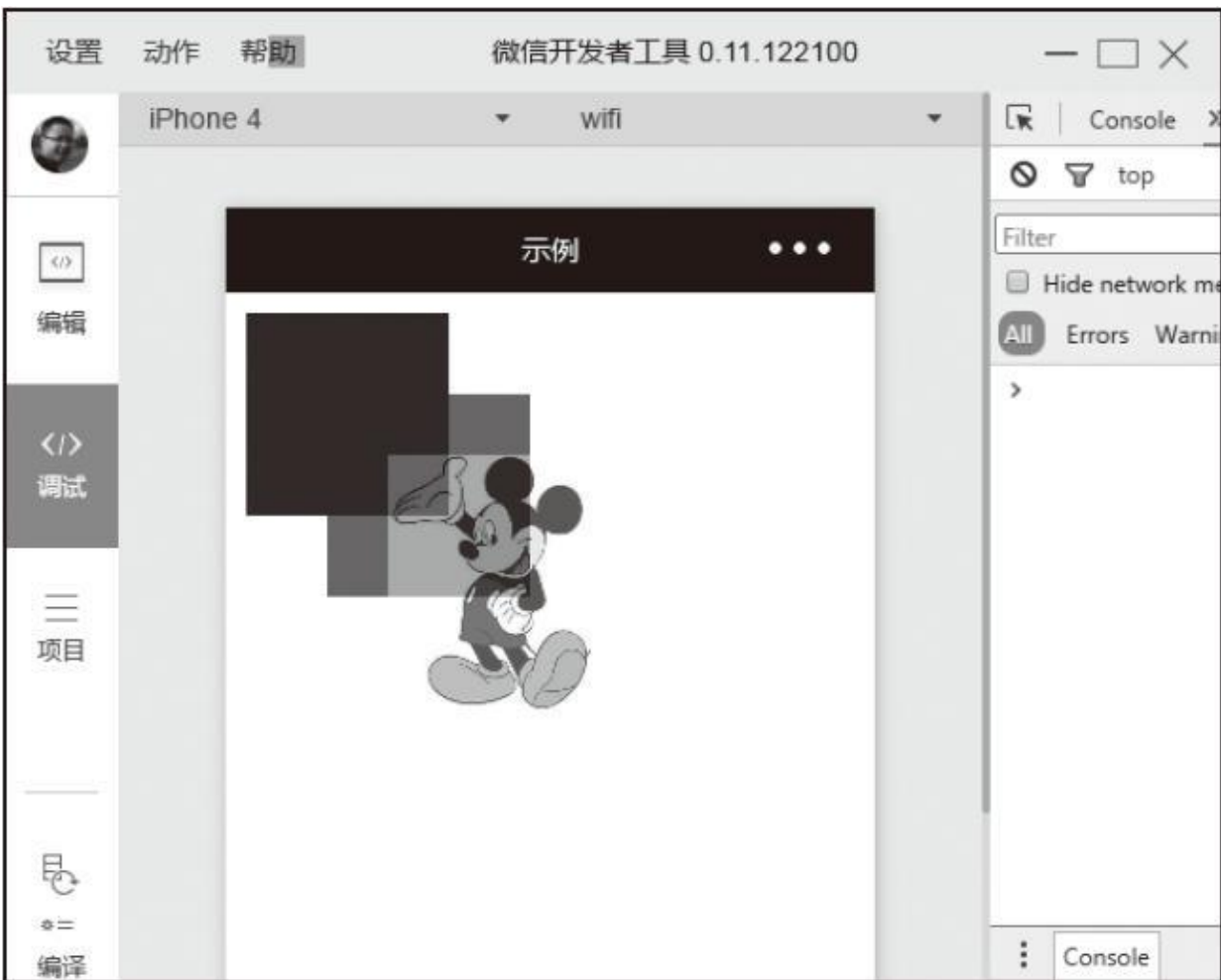


图5-11 混合方法示例

(10) 其他

其余一些方法主要涉及绘画步骤和绘制方式，涉及的方法如下：

·**save ()**：保存当前的绘图上下文，每次保存类似将当前设置进行压栈，可以用于保存一些默认设置，需要注意的是**save ()**方法仅用于保存当前绘图上下文状态，而不是用于保存当前操作步骤。

·**restore ()**：恢复之前保存的绘图上下文。

·**draw (reserve)**：将之前在绘图上下文中的描述（路径、变形、样式）画到**canvas**中。参数包括**reserve**：非必填，指本次绘制是否接着上一次绘制，即**reserve**参数为**false**，则在本次调用**drawCanvas**绘制之前**native**层应先清空画布再继续绘制；若**reserver**参数为**true**，则保留当前画布上的内容，本次调用**drawCanvas**绘制的内容覆盖在上面，默认为**false**。

示例代码如下，效果见图5-12:

```
<canvas style="width: 100%; height: 300px;" canvas-id="myCanvas"></canvas>

Page( {
  onReady : function() {
    var context = wx.createCanvasContext( 'myCanvas' );

    context.setFillStyle( 'red' );
    context.save();
    context.setFillStyle( 'blue' );
    context.fillRect( 10, 10, 100, 100 );
    // 恢复到之前红色设置
    context.restore();
    context.fillRect( 120, 10, 100, 100 );
    context.draw();
  }
} );
```

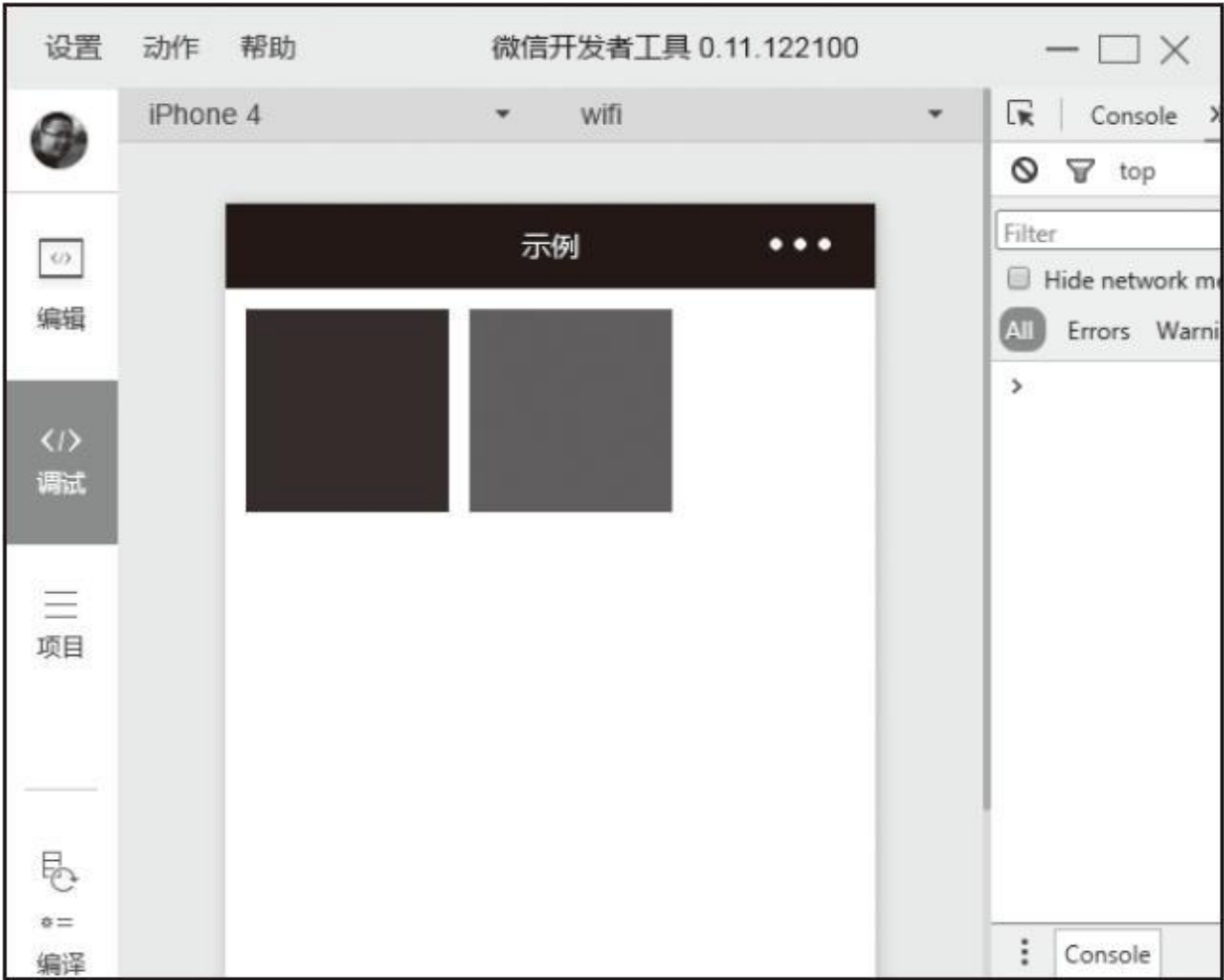


图5-12 其他方法示例

5.7.6 下拉刷新

下拉刷新API包括：

·`onPullDownRefresh`（callback）监听下拉刷新事件需要在Page中定义`onPull-DownRefresh`处理函数。监听下拉事件时需要在app.json配置中开启window配置的`enablePullDownRefresh`选项。当处理完数据刷新后，可以调用`wx.stopPull-DownRefresh`方法停止当前页面的下拉刷新。

·`wx.stopPullDownRefresh`（）停止当前页面下拉刷新。

示例代码如下：

```
Page( {  
  // 监听当前页面下拉刷新  
  onPullDownRefresh : function() {  
    // 停止当前页面下拉刷新  
    wx.stopPullDownRefresh();  
  }  
} );
```

5.8 开放接口

考虑到小程序的登录和支付相对难理解，在第8章我们特意整理一个打赏App，以展示在项目中如何使用登录、支付相关API。

5.8.1 登录

在小程序中，登录分2步，第一步需要获取登录凭证，第二步用登录凭证获取用户登录态信息，登录态信息可用于后续支付等流程。

1.wx.login (Object)

调用接口获取登录凭证（code）进而换取用户登录态信息，包括用户唯一表示（openid）及本次登录的会话密钥（session_key），用户数据的加解密通信需要依赖会话密钥完成，Object参数属性如下：

·success: 接口调用成功的回调函数，返回参数如下：

·errMsg: 调用结果。

·code: 用户允许登录后，回调内容会带上code（有效期五分钟），开发者需要将code发送到开发者服务器后台，使用code换取session_key API，将code换成openid和session_key。

·fail: 接口调用失败的回调函数。

·complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例如下：

```
wx.login( {
  success : function( res ) {
    if ( !res.code ) {
      return;
    }
    // 这里建议调用后台接口进行登录态转换、保存工作
    wx.request({
      url : 'https://myserver.com/login',
      data : {
        code : res.code
      },
      success : function( loginInfo ) {
        console.log( '登录成功' );
      }
    });
  }
});
```

2.code换取session_key，获取用户信息

调用wx.login（）获取code后我们需要在5分钟内用code换取session_key、openid等用户信息，为此官方暴露了一个HTTP接口，尽管我们可以直接通过wx.request（）调用接口，获取用户信息，但是由于session_key是对用户数据进行加密签名的密钥，为了自身应用安全，尽量使用后台服务器调用这个接口，保存登录信息，返回给小程序前台，接口地址如下：https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code

接口请求参数：

·**appid**: 小程序唯一标识，在开发者后台“设置→开发设置”中可找到。

·**secret**: 小程序的app secret，在开发者后台“设置→开发设置”中可找到。

·**js_code**: 调用wx.login () 登录时获取的code。

·**grant_type**: 填写为“authorization_code”。

接口返回参数:

·**openid**: 用户唯一标识。

·**session_key**: 会话密钥。

·**expires_in**: 会话有效期，以秒为单位，例如2592000代表会话有效期为30天。

返回值示例如下:

```
//正常返回的JSON数据包
{
  "openid": "OPENID",
  "session_key": "SESSIONKEY"
  "expires_in": 2592000
}
//错误时返回JSON数据包(示例为Code无效)
{
  "errcode": 40029,
  "errmsg": "invalid code"
}
```

3.登录态维护

开发中，每个项目应该利用后台自己维护登录态，不能直接把 `session_key`、`openid` 等字段作为用户的标识或者 `session` 的标识，具体使用场景可参考第8章。

4.wx.checkSession (Object)

检查登录态是否过期， `Object` 参数属性如下：

- `success`: 接口调用成功的回调函数，登录态未过期。
- `fail`: 接口调用失败的回调函数，登录态已过期。
- `complete`: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例如下：

```
wx.checkSession({
  success: function(){
    //登录态未过期
  },
  fail: function(){
    //登录态过期
  }
})
```

5.8.2 用户信息

wx.getUserInfo (Object)

获取用户信息，需要首先调用wx.login接口，Object参数属性如下：

- success: 接口调用成功的回调函数，success返回参数属性如下：
- userInfo: 用户信息对象，不包含openid等敏感信息。
- rawData: 不包含敏感信息的原始数据字符串，用于计算签名。
- signature: 使用sha1（rawData+sessionkey）得到的字符串，用于校验用户信息。
- encryptData: 包括敏感数据在内的完整用户信息的加密数据。
- iv: 加密算法的初始向量。
- fail: 接口调用失败的回调函数。
- complete: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.getUserInfo({
  success : function( res ) {
    console.log( res ); // 打印出用户信息
  }
});
```

对encryptedData解密后可得到完整的用户信息，解密算法可参考<https://mp.weixin.qq.com/debug/wxadoc/dev/api/signature.html>，解压后的结构如下：

```
{
  "openId": "OPENID",
  "nickName": "NICKNAME",
  "gender": GENDER,
  "city": "CITY",
  "province": "PROVINCE",
  "country": "COUNTRY",
  "avatarUrl": "AVATARURL",
  "unionId": "UNIONID",
  "watermark":
  {
    "appid": "APPID",
    "timestamp": "TIMESTAMP"
  }
}
```

其中unionId用于区分不同应用下的用户身份，它在同一个微信开放平台账号下的移动应用、网站应用和公共账号（包括小程序）是全局唯一的，而openId只在同一应用体系中是唯一的，即在不同公共号下的openId可能重复，但unionId不会重复。即同一用户，对同一个微信开放平台下的不同应用，登录后unionId是相同的，openId是不相同的。获取unionId首先需要将小程序接入微信开放平台，开发者可以登录微信开放平台（open.weixin.qq.com），进入“管理中心->公众账号->

绑定公众帐号”，添加账号。如果是第一次添加需要登录微信开发者平台，进入“账号中心->开发者资质认证”，完成开发者资质认证。

5.8.3 微信支付

本节只做简单的API讲解，支付具体使用方式可参考打赏App。

`wx.requestPayment` (Object) ;

用于发起微信支付，关于支付可参考打赏App，Object参数属性如下：

- timestamp: 时间戳，从1970年1月1日00: 00: 00至今的秒数，即当前的时间。

- nonceStr: 随机字符串，长度为32个字符以下。

- package: 统一下单接口返回的prepay_id参数值，提交格式如：
prepay_id=*

- signType: 签名算法，暂支持MD5。

- paySign: 签名。

- success: 接口调用成功的回调函数。

- fail: 接口调用失败的回调函数。

·**complete**: 接口调用结束的回调函数（调用成功、失败都会执行）。

示例代码如下：

```
wx.requestPayment( {  
  'timeStamp' : '',  
  'nonceStr' : ''  
} );
```

5.8.4 模板消息

对于使用过微信的用户来讲模板消息一点也不陌生，最常见的就是通过微信支付后，微信支付推送的支付消息就是模板消息。在小程序中我们也能使用模板消息给用户推送信息。需要注意的是只有微信6.5.2及以上版本能支持模板功能，低于该版本将无法收到模板消息。

1.调用模板消息接口

发送模板信息需要通过POST方式调用微信后台接口，接口调用可以是前台调用，也可以是后台调用，接口地址为：

[https://api.weixin.qq.com/cgi-bin/message/wxopen/template/send?
access_token=ACCESS_TOKEN](https://api.weixin.qq.com/cgi-bin/message/wxopen/template/send?access_token=ACCESS_TOKEN)

post参数如下：

·touser: 接收者（用户）的openid，通常是当前使用者的openid，必填项。

·template_id: 所需下发的模板消息的id，必填项。

·page: 点击模板卡片后的跳转页面，仅限本小程序内的页面。支持带参数，（示例index? foo=bar）。该字段不填则模板无跳转。

·**form_id**: 表单提交场景下，为submit事件带上的formId；支付场景下，为本次支付的prepay_id，必填项。

·**data**: 模板内容，不填则下发空模板，data中keyword对应模板keyword，必填项。

·**color**: 模板内容字体的颜色，不填默认黑色。

·**emphasis_keyword**: 模板需要放大的关键词，不填则默认无放大。

示例代码如下：

```
{
  "touser": "OPENID",
  "template_id": "TEMPLATE_ID",
  "page": "index",
  "form_id": "FORMID",
  "data": {
    "keyword1": {
      "value": "薯片",
      "color": "#173177"
    },
    "keyword2": {
      "value": "2015年01月05日 12:30",
      "color": "#173177"
    }
  },
  "emphasis_keyword": "keyword1.DATA"
}
```

接口返回数据：

```
{"errcode": 0, "errmsg": "ok"}
```

返回数据中常见错误码有：

- 40037: `template_id`不正确。
- 41028: `form_id`不正确, 或者过期。
- 41029: `form_id`已被使用。
- 41030: `page`不正确。
- 45009: 接口调用超过限额 (目前默认每个帐号日调用限额为100万)。

调用模板消息时有几个参数非常关键, 分别是: `access_token` (接口url参数), `form_id`, `template_id`和`data`。接下来我们一一分析这些参数。

(1) `template_id`和`data`

`template_id`是需要调用模板的id, `data`为当前模板所需要的数据, 小程序所有模板都在微信公共平台 (<https://mp.weixin.qq.com/>) ->模板消息进行管理, `template_id`可在模板管理界面中直接复制, 如图5-13所示。



图5-13 获取template_id

没有模板时可以创建一个新模板，微信定制了多种类型的模板，每个模板可以选取需要填写哪些key值，这些key值会根据请求参数data属性进行填写，选取好后会生成相应的模板如图5-14所示。



图5-14 模板详情

当前模板按上文示例代码请求数据会给用户推送一条物品名称为“薯片”，付款时间为“2015年01月05日12: 30”的付款成功通知消息。

(2) form_id

form_id的值由页面<form/>组件submit方法生成，获取时需要将<form/>的report-submit属性值为true，此时点击提交按钮触发submit事件时，可通过参数获取。

示例代码如下：

```
<form report-submit bindsuit="submit">
  <button form-type="submit">提交</button>
</form>

Page( {
  submit : function( e ) {
    console.log( e.detail.formId );
  }
} );
```

当用户完成支付行为时，form_id的值应为统一下单接口返回的prepay_id。

(3) access_token

access_token是全局唯一的接口调用凭证，调用很多接口都需要使用access_token。access_token存储至少需要512个字符空间，获取access_token需要通过get方式调用微信后台接口：

<https://api.weixin.qq.com/cgi-bin/token?>

[grant_type=client_credential&appid=APPID&secret=APPSECRET](https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET)

其参数如下：

·grant_type: 接口类型，获取access_token时填写client_credential，必填项。

·appid: 第三方用户唯一凭证，即小程序appid，可在微信公众平台官网->设置->开发设置中获取，必填项。

·secret: 第三方用户唯一凭证密钥，即appsecret，可在微信公众平台官网->设置->开发设置中获取，必填项。

正常情况下，微信会返回如下JSON数据：

```
{"access_token": "ACCESS_TOKEN", "expires_in": 7200}
```

返回数据中expires_in指凭证有效时间，单位秒。由此可看出access_token有效时间为2小时，失效后需要定时刷新，重复刷新将导致上次获取的access_token失效，同时access_token有失效时间而且每天调用接口次数有限制，如果每个服务（或小程序客户端）单独调用接口获取access_token将会导致access_token不一致，产生冲突，导致服务不稳定，所以我们通常利用中控服务器单独维护access_token，系统所有服务都依赖这个服务获取access_token，这样能保证access_token的一致性和有效生命周期，获取access_token方式如下：

·为了保密appsecret，第三方需要一个access_token获取和刷新的中控服务器。而其他业务逻辑服务器所使用的access_token均来自于该中

控服务器，不应该各自去刷新，否则会造成access_token覆盖而影响业务；

- 目前access_token的有效期通过返回的expires_in来传达，目前是7200秒之内的值。中控服务器需要根据这个有效时间提前去刷新新access_token。在刷新过程中，中控服务器对外输出的依然是老access_token，此时公众平台后台会保证在刷新短时间内，新老access_token都可用，这保证了第三方业务的平滑过渡；

- access_token的有效时间可能会在未来有调整，所以中控服务器不仅需要内部定时主动刷新，还需要提供被动刷新access_token的接口，这样便于业务服务器在API调用获知access_token已超时的情况下，可以触发access_token的刷新流程。

2. 下发条件

调用模板信息需要满足一些相应条件，按类型可将这些条件分为支付和提交表单：

- 支付当用户在小程序内完成过支付行为，可允许开发者向用户在7天内推送有限条数的模板消息（1次支付可下发1条，多次支付下发条数独立，互相不影响）。

·提交表单当用户在小程序内发生过提交表单行为且该表单声明为要发模板消息的，开发者需要向用户提供服务时，可允许开发者向用户在7天内推送有限条数的模板消息（1次提交表单可下发1条，多次提交下发条数独立，相互不影响）。

3.模板管理规则

（1）模板审核

新建的模板需要通过审核才能使用，审核规则如下：

标题：

- 1) 标题不能存在相同。
- 2) 标题意思不能存在过度相似。
- 3) 标题必须以“提醒”或“通知”结尾。
- 4) 标题不能带特殊符号、个性化字词等没有行业通用性的内容。
- 5) 标题必须能体现具体服务场景。
- 6) 标题不能涉及营销相关内容，包括但不限于：消费优惠类、购物返利类、商品更新类、优惠券类、代金券类、红包类、会员卡类、积分分类、活动类等营销倾向通知。

关键词：

- 1) 同一标题下，关键词不能存在相同。
- 2) 同一标题下，关键词不能存在过度相似。
- 3) 关键词不能带特殊符号、个性化字词等没有行业通用性的内容。
- 4) 关键词内容示例必须与关键词对应匹配。
- 5) 关键词不能太过宽泛，需要具有限制性，例如：“内容”这个就太宽泛，不能审核通过。

(2) 违规说明

除不能违反运营规范外，还不能违反以下规则，包括但不限于：

- 1) 不允许恶意诱导用户进行触发操作，以达到可向用户下发模板目的。
- 2) 不允许恶意骚扰，下发对用户造成骚扰的模板。
- 3) 不允许恶意营销，下发营销目的模板。
- 4) 不允许通过服务号下发模板来告知用户在小程序内触发的服务相关内容处罚说明。

(3) 处罚说明

根据违规情况给予相应梯度的处罚，一般处罚规则如下：

- 1) 第一次违规，删除违规模板以示警告，
- 2) 第二次违规，封禁接口7天。
- 3) 第三次违规，封禁接口30天。
- 4) 第四次违规，永久封禁接口。

处罚结果及原因以站内信形式告知。

5.8.5 客服消息

在小程序中，通过点击<contact-button/>可以进入客服会话系统。在会话系统中，用户发送的信息（或进行某些特定的用户操作引发的事件推送），微信服务器会将消息（或事件）的数据包以POST方式传递给开发者填写的URL。当开发者服务器接受到信息后，可以使用发送服务消息接口进行异步回复，这两个行为分别是2个不同的请求。基于这个特性，可以自己研发或接入任何客服系统。

1.接入指引

接入微信小程序消息服务，整体需要2步：

- 1) 填写开发者服务器配置。
- 2) 开发者服务器接受请求并返回指定信息，验证服务器地址有效性。

(1) 填写服务器配置

登录微信公众平台，在“设置->消息推送”中启动消息服务，如图5-15所示。在配置页面填写URL、Token和EncodingAESKey。其中URL是开发者服务端接收消息的接口URL地址，必须是http://或https://开头，分别支持80和443端口。Token为用户身份令牌用作生成签名，可由开

发者任意填写，该Token会和接口URL中包含的Token进行对比，从而验证安全性。EncodingAESKey由开发者手动填写或随机生成，用作消息体加解密密钥。在配置界面中可以选择消息加密方式和数据格式，加密方式默认为明文，数据格式默认为XML。消息加密解密可参考https://open.weixin.qq.com/cgi-bin/showdocument?action=dir_list&t=resource/res_list&verify=1&id=open1419318479&token=&lang=zh_CN。

消息推送配置

填写的URL需要正确响应微信发送的Token验证，填写说明请阅读消息推送服务器配置指南

URL(服务器地址)

必须以http://或https://开头，分别支持80端口和443端口

Token(令牌)

必须为英文或数字，长度为3-32字符

EncodingAESKey
(消息加密密钥)

0/43

随机生成

消息加密密钥由43位字符组成，字符范围为A-Z,a-z,0-9

消息加密方式

请根据业务需要，选择消息加密方式，启用后将立即生效

☒ 明文模式

(不使用消息体加解密功能，安全系数较低)

☐ 兼容模式

(明文、密文将共存，方便开发者调试和维护)

☐ 安全模式 (推荐)

(消息包为纯密文，需要开发者加密和解密，安全系数高)

数据格式

请选择微信与开发者服务器间传输的数据格式

☐ JSON

☒ XML

提交

图5-15 配置界面

(2) 验证服务器地址有效性

仅仅填写配置信息是不能完成接入的，提交配置信息后，微信服务器将发送GET请求到填写的服务器地址，开发者服务器收到请求并按要求返回才能完成接入，所以我们在配置前先完成后台服务器接口工作，GET请求参数如下：

- signature: 微信加密签名，signature结合了开发者填写的token参数和请求中的timestamp参数、nonce参数。

- timestamp: 时间戳。

- nonce: 随机数。

- echostr: 随机字符串。

后台可通过检验signature验证请求。收到GET请求后，开发者服务器需要原样返回echostr参数内容，才能接入生效。加密/校验流程如下：

- 1) 将token、timestamp、nonce三个参数进行字典序排序。
- 2) 将三个参数字符串拼接成一个字符串进行sha1加密。

3) 开发者获得加密后的字符串可与signature对比，标识该请求来源于微信。

检验signature的PHP代码示例如下：

```
private function checkSignature()
{
    $signature = $_GET["signature"];
    $timestamp = $_GET["timestamp"];
    $nonce = $_GET["nonce"];

    $token = TOKEN;
    $tmpArr = array($token, $timestamp, $nonce);
    sort($tmpArr, SORT_STRING);
    $tmpStr = implode( $tmpArr );
    $tmpStr = sha1( $tmpStr );

    if( $tmpStr == $signature ){
        return true;
    }else{
        return false;
    }
}
```

完成URL有效性验证后便接入生效。通过配置URL接口，开发者服务器可接受微信服务器推送过来的消息和事件，并根据业务进行响应。

2.接受消息和事件

当点击<contact-button/>时便进入了客服会话状态，在会话状态中开发者服务器接收信息和发送信息是2个不同接口。当用户在客服会话中发送消息（或进行某些特定的用户操作行为引发的事件），微信服务器会将消息（或事件）的数据包（JSON或XML格式）以POST方式发送到开发者填写的URL。

接收到信息后开发者服务器必须做出下述回复，告知微信服务器我方服务器已成功接收信息：

- 1) 直接回复**success**（推荐方式）。
- 2) 直接回复空串（指字节长度为0的空字符串，而不是结构体中**content**字段的内容为空）。

微信服务器在5秒内收不到任何响应会断掉连接，并重新发起请求，总共重试3次，如果在调试中，发现用户无法收到响应的消息，可以检查是否消息处理超时。关于重试的消息排重，有**msgid**的消息推荐使用**msgid**排重。事件类型消息推荐使用**FromUserName+CreateTime**排重。一旦遇到以下情况，微信都会在小程序会话中，向用户发出系统提示“该小程序客服暂时无法提供服务，请稍后再试”：

- 1) 开发者在5秒内未回复任何内容。
- 2) 开发者回复了异常数据。

如果希望增加消息安全性，可以在消息配置中开启消息加密功能，用户发给小程序的消息以及小程序被动回复用户消息都会继续加密。接收的消息按类型可划分为3类：进入会话事件、文本消息、图片消息。

(1) 进入会话事件

当用户点击小程序“客服会话按钮”进入客服会话时将产生如下数据包:

XML格式:

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>1482048670</CreateTime>
  <MsgType><![CDATA[event]]></MsgType>
  <Event><![CDATA[user_enter_tempsession]]></Event>
  <SessionFrom><![CDATA[sessionFrom]]></SessionFrom>
</xml>
```

JSON格式:

```
{
  "ToUserName": "toUser",
  "FromUserName": "fromUser",
  "CreateTime": 1482048670,
  "MsgType": "event",
  "Event": "user_enter_tempsession",
  "SessionFrom": "sessionFrom"
}
```

字段说明如下:

- ToUserName: 小程序的原始ID。
- FromUserName: 发送者的openid。
- CreateTime: 事件创建时间（整型）。
- MsgType: event。
- Event: 事件类型，user_enter_tempsession。

·SessionFrom: 开发者在客服会话按钮设置的sessionFrom参数。

(2) 文本消息

用户在客服会话中发送文本消息时将产生如下数据包:

XML格式:

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>1482048670</CreateTime>
  <MsgType><![CDATA[text]]></MsgType>
  <Content><![CDATA[this is a test]]></Content>
  <MsgId>1234567890123456</MsgId>
</xml>
```

JSON格式:

```
{
  "ToUserName": "toUser",
  "FromUserName": "fromUser",
  "CreateTime": 1482048670,
  "MsgType": "text",
  "Content": "this is a test",
  "MsgId": 1234567890123456
}
```

字段说明如下:

·ToUserName: 小程序的原始ID。

·FromUserName: 发送者的openid。

·CreateTime: 消息创建时间（整型）。

- MsgType: text 。
- Content: 文本消息内容 。
- MsgId: 消息id, 64位整型 。

(3) 图片消息

用户在客服会话中发送图片消息时将产生如下数据包:

XML格式:

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>1482048670</CreateTime>
  <MsgType><![CDATA[image]]></MsgType>
  <PicUrl><![CDATA[this is a url]]></PicUrl>
  <MediaId><![CDATA[media_id]]></MediaId>
  <MsgId>1234567890123456</MsgId>
</xml>
```

JSON格式:

```
{
  "ToUserName": "toUser",
  "FromUserName": "fromUser",
  "CreateTime": 1482048670,
  "MsgType": "image",
  "PicUrl": "this is a url",
  "MediaId": "media_id",
  "MsgId": 1234567890123456
}
```

字段说明如下:

- ToUserName: 小程序的原始ID 。

- FromUserName: 发送者的openid。
- CreateTime: 消息创建时间（整型）。
- MsgType: text。
- PicUrl: 图片链接（由系统生成）。
- MediaId: 图片消息媒体id，可以调用获取临时素材接口拉取数据。
- MsgId: 消息id，64位整型。

3.发送消息

开发者在接收到微信服务器消息后，在一段时间内（目前修改为48小时）可以调用客服接口，通过POST一个JSON数据包来给用户发送消息。此接口主要用户客服等有人工消息处理环节，方便开发者为用户提供更优质的服务。

目前允许的动作列表如下，不同动作触发后，允许的客服接口下发消息条数和下发时限不同。下发条数达到上限后，会收到错误返回码，具体请见返回码说明页：

用户动作	允许下发条数限制	下发时限
用户通过客服消息按钮进入会话	1 条	1 分钟
用户发送信息	3 条	48 小时

发送消息直接通过POST方式调用一下接口即可：

<https://api.weixin.qq.com/cgi-bin/message/custom/send?>

[access_token=ACCESS_TOKEN](#)

消息JSON数据格式如下：

文本消息：

```
{
  "touser": "OPENID",
  "msgtype": "text",
  "text": {
    "content": "Hello World"
  }
}
```

图片消息：

```
{
  "touser": "OPENID",
  "msgtype": "image",
  "image": {
    "media_id": "MEDIA_ID"
  }
}
```

字段说明如下：

·access_token：调用接口凭证，必填项。

·touser：普通用户openid，必填项。

·msgtype：消息类型，文本为text，图片为image，必填项。

·content: 文本消息内容，必填项。

·media_id: 发送的图片的媒体ID，通过新增素材接口上传图片文件获得，必填项。

请求接口后微信服务器将返回调用结果，其返回错误码意义如下：

·-1: 系统繁忙，此时请开发者稍候再试。

·0: 请求成功。

·40001: 获取access_token时AppSecret错误，或者access_token无效。请开发者认真比对AppSecret的正确性，或查看是否正在为恰当的小程序调用接口。

·40002: 不合法的凭证类型。

·40003: 不合法的OpenID，请开发者确认OpenID否是其他小程序的OpenID。

·45015: 回复时间超过限制。

·45047: 客服接口下行条数超过上限。

·48001: API功能未授权，请确认小程序已获得该接口。

4.临时素材接口

在发送消息时，`media_id`需要填写临时素材id。临时素材需要先调用新增临时素材接口上传至微信服务，在通过获取接口获取对应资源id。目前小程序只支持下载图片文件。

(1) 新增临时素材

通过新增素材接口可以把本地媒体文件（目前仅支持图片）上传至微信服务器，用于用户发送客服消息或被动回复用户消息。接口请求为POST/FORM，调用接口如下：

[https://api.weixin.qq.com/cgi-bin/media/upload?
access_token=ACCESS_TOKEN&type=TYPE](https://api.weixin.qq.com/cgi-bin/media/upload?access_token=ACCESS_TOKEN&type=TYPE)

参数说明如下：

- `access_token`: 调用接口凭证，必填项。
- `type`: image，必填项。
- `media`: form-data中媒体文件标识，有filename、filelength、content-type等信息。

示例代码，利用curl命令，用FORM表单方式上传一个多媒体文件：

```
curl -F media=@test.jpg https://api.weixin.qq.com/cgi-bin/media/upload?
access_token=ACCESS_TOKEN&type=TYPE
```

调用后正常情况会返回以下JSON数据:

```
{
  "type": "TYPE",
  "media_id": "MEDIA_ID",
  "created_at": 123456789
}
```

错误情况会返回以下JSON数据:

```
{
  "errcode": 40004,
  "errmsg": "invalid media type"
}
```

返回字段如下:

- type: image。
- media_id: 媒体上传后, 获取标识。
- created_at: 媒体文件上传时间戳。

(2) 获取临时素材

通过调用获取临时素材接口, 可以获取已上传的临时文件, 目前小程序仅支持下载图片文件。接口请求为**POST/FORM**, 调用接口如下:

<https://api.weixin.qq.com/cgi-bin/media/get?>

[access_token=ACCESS_TOKEN&media_id=MEDIA_ID](#)

参数说明如下:

·access_token: 调用接口凭证。

·media_id: 媒体文件ID。

示例代码, 利用curl命令获取多媒体文件:

```
curl -I -G "https://api.weixin.qq.com/cgi-bin/media/get?
access_token=ACCESS_TOKEN&media_id=MEDIA_ID"
```

调用成功后, 正常情况将返回HTTP头如下:

```
HTTP/1.1 200 OK
Connection: close
Content-Type: image/jpeg
Content-disposition: attachment; filename="MEDIA_ID.jpg"
Date: Sun, 06 Jan 2013 10:20:18 GMT
Cache-Control: no-cache, must-revalidate
Content-Length: 339721
```

如果返回的是视频素材, 内容如下:

```
{ "video_url":DOWN_URL}
```

错误情况将返回一下JSON:

```
{
  "errcode":40007,
```

```
    "errmsg": "invalid media_id"  
  }
```

5.8.6 分享

小程序分享页面需要在Page中定义onShareAppMessage函数，设置该页面分享信息。只有定义了此事件函数，右上角才会显示“分享”按钮。用户点击分享按钮时会触发该函数，该函数返回的Object对象将用于定义分享内容，目前分享图片不能自定义，系统会取当前页面，从顶部开始，高度为80%屏幕宽度的图像作为分享图片。返回Object属性如下：

- title: 分享标题，默认值为当前小程序名称。
- desc: 分享描述，默认值为当前小程序名称。
- path: 分享路径，必须是以/开头且在app.json中已注册的路径，默认为当前页面path。

示例代码如下：

```
Page({
  onShareAppMessage: function () {
    return {
      title: '自定义分享标题',
      desc: '自定义分享描述',
      path: '/page/home?tab=cart'
    }
  }
})
```

5.8.7 获取二维码

小程序除了分享当前页面也可以通过调用后台接口生成任意页面
对应二维码，扫描该二维码即可直接进入小程序对应页面。通过该接
口，仅能生成已发布小程序的二维码。在开发过程中可在开发者工具
预览时生成开发版的带参二维码。带参二维码只有10000个，请谨慎使
用。接口请求方式为**POST**，接口地址如下：

[https://api.weixin.qq.com/cgi-bin/wxaapp/createwxaqrcode?
access_token=ACCESS_TOKEN](https://api.weixin.qq.com/cgi-bin/wxaapp/createwxaqrcode?access_token=ACCESS_TOKEN)

接口中**access_token**为全局唯一调用凭据，可参考模板消息调用，
其余**POST**参数如下：

·**path**： 页面路径，不能为空，最大长度128字节。

·**width**： 二维码的宽度，默认值为430。

5.9 小结

本章主要介绍小程序**API**，无论是组件还是**API**，大家都可以当做字典查询，平时大概知道它们能提供的能力、边界，在需要用到的时候再细致研究。目前小程序组件和**API**都在不断丰富完善中，平时多关注官方信息。在接下来章节中我们将进入实战环节。

第6章 案例分析——豆瓣电影

在前5章中，我们学习了小程序框架、组件、API以及开发相关的基础知识。从本章开始，我们将进入实战环节，开发小型App。现阶段小程序还不太成熟，在不同手机、不同版本微信环境下存在一些差异，实战实例中我们主要讲解项目的架构思路和实现过程，通过学习，再看官方文档会变得非常容易。通过实践项目的学习，大家会对小程序研发有更深刻的认识，为了便于学习，实战项目中均未使用ES6新特性，本章的所有源码可在<https://github.com/wxapp-book/douban.git> 下载。

为了让大家快速了解开发流程，我们选用豆瓣电影作为第一个实例，在这个实例中，我们主要调用豆瓣的开放API接口，这样我们能够将更多精力集中到小程序本身的开发上。豆瓣对接口作了权限设置，公开的数据相对较少，这使得我们的实践项目只有两个页面。电影列表页和电影详情页（如图6-1所示），虽然项目小、代码量不大，但是在架构上本项目是前端常用架构，基于这套代码架构可以搭建任意复杂的项目。

6.1 准备工作

在开始代码编写前，首先确认服务端域名已添加在微信受信任链接中，同时是HTTPS服务。

6.1.1 豆瓣API

现在很多公司都有自己的开放平台，比如淘宝开放平台、腾讯开发者平台、高德API、百度API等，它们都提供了很多强大的平台数据服务，利用这些服务我们能很快搭建出应用，提高开发效率、节省服务器资源。这些API大部分都是需要申请的，豆瓣API在部分信息上也作了权限限制，我们能调用部分接口和接口中部分数据。本实例仅用到豆瓣电影API，我们可以在豆瓣API官网

（http://developers.douban.com/wiki/?title=api_v2）上看到所有API信息，目前API是2.0版本，提供的服务如图6-2所示。



图6-1 豆瓣电影界面

Api V2 索引
图书Api V2
电影Api V2
音乐Api V2
同城Api V2
广播Api V2
用户Api V2
日记Api V2
相册Api V2
线上活动Api V2
论坛Api V2
回复Api V2
我去Api V2

图6-2 豆瓣API V2

进入电影API，我们主要使用正在热映、即将上映、电影条目信息这3个API接口：

·正在热映

接口地址：https://api.douban.com/v2/movie/in_theaters

参数city: 正在热映列表地址, 可以传中文字符或城市编号, 如“北京”、“110000”, 默认为北京。

例: https://api.douban.com/v2/movie/in_theaters? city=北京

·即将上映

接口地址: https://api.douban.com/v2/movie/coming_soon

参数如下:

·start: 开始的节点, 默认为0。

·count: 请求返回列表条数, 默认为20。

例: https://api.douban.com/v2/movie/in_theaters? start=0&count=20

·电影条目信息

接口地址: <https://api.douban.com/v2/movie/subject/: id>, 这个接口没有参数

例: <https://api.douban.com/v2/movie/subject/1324043>

6.1.2 跳转层

由于豆瓣API不能支持非浏览器客户端发起的请求，非浏览器客户端需要下载对应的SDK来请求API，所以小程序在微信中打开时不能直接调用豆瓣API接口，所以我们利用自己的服务器做了一次请求跳转（如图6-3所示），这也是无奈之举：

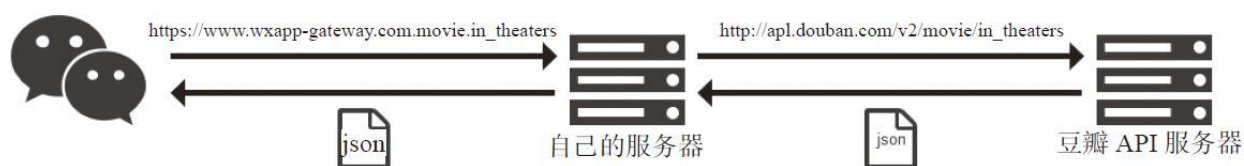


图6-3 服务器请求流程

豆瓣API非常规范，后台只需要做简单跳转便可以实现跳转，当用户请求https://www.wxapp-gateway.com/movie/in_theaters 时，我们后台服务请求豆瓣对应的API地址：

http://api.douban.com/v2/movie/in_theaters，服务端我们使用

nodejs+express实现后台，搭建nodejs和express的过程我们不再赘述，大家可以参考网络教程，具体router跳转代码如下：

```
var express = require('express'),
    router = express.Router(),
    http = require( 'http' ),
    _fn;

/* GET users listing. */
router.get('/', function(req, res, next) {
  // 拦截所有movie子域下的请求，获取真正的请求地址
  var path = req.originalUrl.replace( '/movie/', '' );
  _fn.getData( path, function( data ) {
    // 返回从豆瓣请求的数据
  })
})
```

```

        res.send( data );
    } });
});

_fn = {
  getData : function( path, callback ) {
    // 向豆瓣发起请求
    http.get( {
      hostname : 'api.douban.com',
      path : '/v2/movie/' + path
    }, function( res ) {
      var body = [];
      // 监听chunk包
      res.on( 'data', function( chunk ) {
        body.push( chunk );
      } );
      // 回调函数
      res.on( 'end', function( ) {
        // 将buffer转为string,并回调方法
        body = Buffer.concat( body );
        callback( body.toString() );
      } );
    } );
  }
}

module.exports = router;

...
var movie = require('./routes/movie');
var app = express();
// 在app.js中配置路由
app.use('/movie/*', movie);
...

```

6.2 技术架构

在项目编码之前，我们需要先思考项目的架构，保证项目具备一定的拓展性、项目之间的耦合度足够低、项目的代码具备复用的能力，这样才能提高团队的编码效率。借用服务端三层架构模式，一个项目可分为表现层、业务逻辑层、数据访问层，前端项目的架构如图6-4所示。

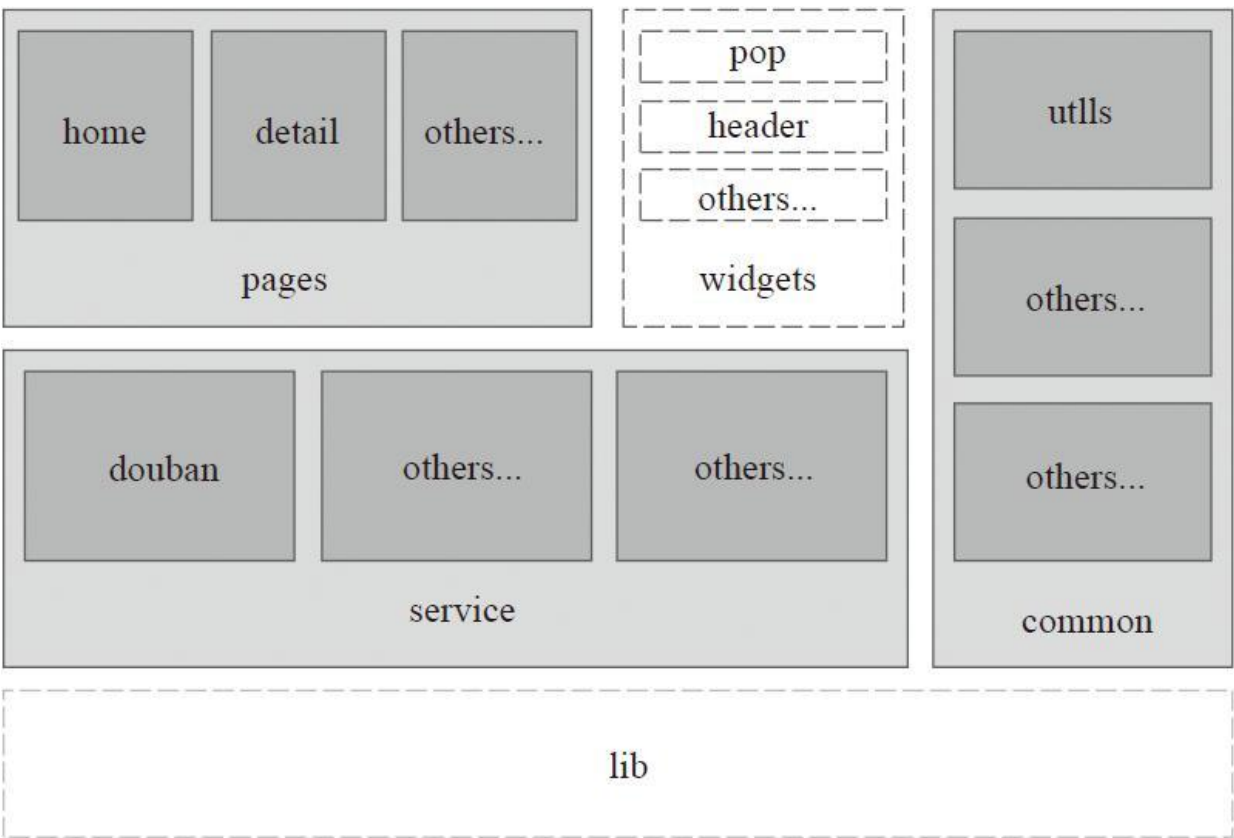


图6-4 豆瓣App架构

·lib: 放置一些最底层、第三方库, 如jquery、seajs、qrcode、echarts等, 通常lib层是全公司共用一套。豆瓣电影中由于没有第三方库文件, 或者说小程序框架本身就是一个第三方库, 所以没有lib层。

·common: 放置和项目相关的一些公共代码, 如转码、工具包、公共样式设置等。根据业务需要可以将widgets和common合并为一个目录。

·service: 业务逻辑层, 按业务类型整合相关的方法, 向上暴露需要的接口方法, 比如a页面需要调用登录和列表信息, b页面需要调用登录和详情信息, 那么这两个页面中的登录接口就会被调用两次, 这时我们便可以将这种和业务紧密相关的接口调用方法封装起来, 作为一个服务暴露给上层应用。业务逻辑层能降低表现层对业务的关注, 让更多精力关注在页面渲染、页面交互等功能上。

·widgets: 一些通用的带UI的小组件, 如pop、header、购物车图标等等, 它和common层有些类似, 但是相对common层, widgets具备UI界面和一个闭环的交互功能。由于业务问题, 在豆瓣电影实例中没有widgets目录。

·pages: 表现层, 一般表现层中一个文件夹对应一个页面所涉及的所有资源。

这种分层方式是前端项目常见的架构分层方式，适用于任何原则，同时按照组件化原则我们通常会把一个组件、包、页面所涉及的所有资源放置在一个目录中，如图6-5所示。

- ▼ demo
 - ▼ common
 - ▼ basestyle
 - ▶ images
 - basestyle.wxss
 - ▼ utils
 - utils.js
 - ▼ pages
 - ▼ detail
 - detail.js
 - detail.json
 - detail.wxml
 - detail.wxss
 - ▼ home
 - home.js
 - home.json
 - home.wxml
 - home.wxss
 - ▼ service
 - ▶ douban
 - app.js
 - app.json
 - app.wxss

图6-5 目录结构

如在**basestyle**中，我们放置了**wxss**和相关的图片资源，这样在不同项目开发中需要复用组件的情况下，我们就可以直接将目录拷贝过去，而不用在不同目录中查找当前组件相关的资源。我们平时编码过程中，尽量把相关的文件放到一个目录中，通常目录名和模块入口名一致。豆瓣电影相关目录如下：

- basestyle**: 一些项目公共的基本样式，被**app.wxss**引入，当前项目中没有公共样式，内容为空。

- utils**: 基础工具包，项目中我们把系统**toast**进行了一层代理，封装到对应方法中，如果以后需要改变**toast**行为，我们便可以直接修改代码，而不用在每个调用页面进行修改。

- detail**: 详情页相关资源。

- home**: 首页相关资源。

- douban**: 请求豆瓣接口的相关逻辑代码。

6.3 公共模块开发

按照上述架构描述，我们对相关的模块和功能进行了划分，现在只需要按照规划实现代码即可。我们首先讲述service和utils的编写。在JavaScript整体编码中一般handle是需要对外暴露的公共方法的句柄，写在模块前面，_fn是私有方法，写在后面，这样便于他人阅读代码，一打开代码便看见当前模块对外暴露的方法，然后层层往下阅读，这是多年形成的编程习惯，大家阅读代码时可以留意下。

6.3.1 业务逻辑层

在本实例中我们将豆瓣相关业务封装到douban.js中，虽然douban.js只有一个文件，按规则我们仍然将它放在一个douban文件夹中，这样做的原因是如果以后douban.js内容过多或需要多人并行开发时，我们可以通过模块化对这个js进行拆分，douban.js代码见代码清单6-1。

代码清单6-1 douban.js

```
var handle,
    URL,
    LISTTYPE,
    _fn;

URL = {
  // 电影列表
  movieList : 'https://www.wxapp-gateway.com/movie/',
  // 电影详情
  movieDetail : 'https://www.wxapp-gateway.com/movie/subject'
}

// 电影类型
LISTTYPE = {
  // 正在热映
  1 : 'in_theaters',
  // 即将上映
  2 : 'coming_soon'
}

// 暴露出去的服务接口
handle = {
  // 获取列表
  getMovieList : function( type, callback ) {
    var url = URL.movieList + LISTTYPE[type];
    _fn.getData( {
      url : url
    }, callback );
  },
  // 获取详情
  getMovieDetail : function( id, callback ){
    var url = URL.movieDetail + '/' + id;
    _fn.getData( {
      url : url
    }
```

```
    }, callback );
  }
}

fn = {
  // 将网络请求统一放置在一个方法中
  getData : function( param, callback ) {
    wx.request( {
      url : param.url,
      method : 'get',
      data : {
        apikey : '00fa6c0654689a0202ef4412fd39ce06'
      },
      header: {
        'Content-Type': 'application/json'
      },
      success : function( e ) {
        callback( e.data );
      },
      fail : function( e ) {
        callback();
      }
    } );
  }
}

module.exports = handle;
```

6.3.2 公共模块

公共模块中我们只有**basestyle**和**utils**两个文件夹，**basesytle**是公共样式，本实例没有实现，按**basestyle**思路我们可以创建其他风格样式，在页面中如果需要切换时直接切换**class**属性名便可以统一切换风格。

utils只是为了给大家进行演示，仅做了简单封装，代码如下：

```
var handle;

handle = {
  showLoading : function () {
    wx.showToast( {
      title : '数据加载中',
      icon : 'loading',
      duration : 10000
    } );
  },
  hideLoading : function() {
    wx.hideToast();
  }
};
```

common目录内的模块并没有统一的拆分规则，可以按实际项目需要进行拆分。

6.4 页面构建

与开发前端项目一样，我们先编写页面构建，再编写逻辑代码。
电影票业务比较简单只有两个页面，即首页和详情页。

6.4.1 首页

构建首页时，我们使用了一个fixed布局的头，这样在视觉上微信原生的头部变高了，在样式效果上，基本没有使用背景图片，通过box-shadow、text-shadow以及背景渐变实现多种效果。列表布局上主要使用了Flex布局，整体结构如图6-6所示。

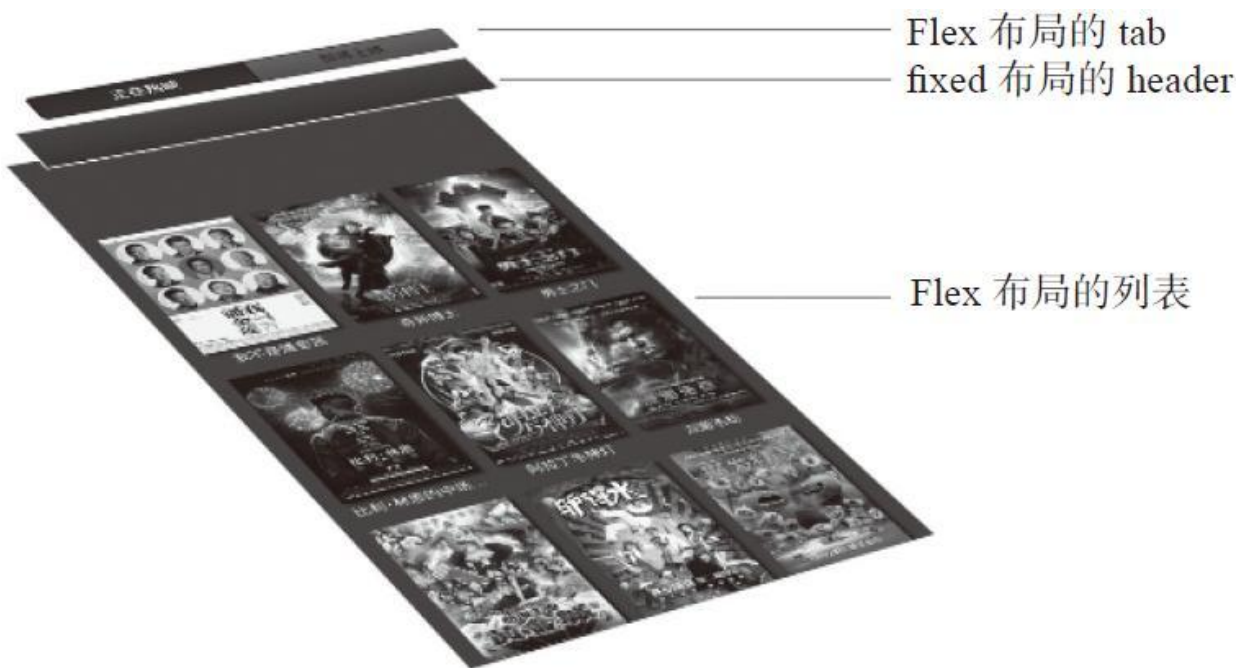


图6-6 首页结构图

首页布局代码如代码清单6-2和代码清单6-3所示。

代码清单6-2 home.wxml

```
<view class="header">
  <view class="nav" bindtap="changeTab">
```

```

        <view class="nav-wrapper">
            <block wx:for="{{tabs.list}}">
                <view class="tab current" data-type="{{item.type}}" data-index="{{index}}"
wx:if="{{index / 1 == tabs.currentIndex / 1}}">{{item.text}}</view>
                <view class="tab" data-type="{{item.type}}" data-index="{{index}}" wx:else>
{{item.text}}</view>
            </block>
        </view>
    </view>
</view>

<view class="list basestyle">
    <block wx:for="{{movies.subjects}}">
        <view class="item">
            <navigator url="../detail/detail?id={{item.id}}">
                <image mode="aspectFill" src="{{item.images.large}}"></image>
                <text>{{item.title}}</text>
            </navigator>
        </view>
    </block>
</view>

```

代码清单6-3 home.wxss

```

/* 头部 */
.header { position: fixed; top : 0; left : 0; padding : 10rpx 30rpx 15rpx 30rpx;
width : 691rpx; background: -webkit-gradient(linear, 0 0, 0 100%,
from(#3f3f3f), to(#2f2f2f)); border-bottom : solid 1px #000; box-shadow:
0 2px 4px #111; }
.header .bar { height : 60rpx; font-size : 28rpx; background-color : #888;
color : #fff; }

/* 导航 */
.nav { height : 60rpx; width : 100%; background-color: #373737; border : solid
1px #000; border-radius: 5px; overflow: hidden; box-shadow : 0 -1px 0px
#777; }
.nav .nav-wrapper { width : 693rpx; height : 100%; display : flex; position:
relative; }
.nav .tab { color : #111; flex-grow : 1; font-size : 28rpx; border-right : solid
1px #000; width : 50%; background: -webkit-gradient(linear, 0 0, 0 100%,
from(#636363), to(#4f4f4f)); text-align: center; line-height: 60rpx; }
.nav .tab:first{ border-left : none; }
.nav .tab.current { color : #fcfcfc; font-weight : bold; background:
-webkit-gradient(linear, 0 0, 0 100%, from(#252525), to(#2f2f2f));
box-shadow: inset 0 0 4px #111; text-shadow : 0 2px 3px #000; }

/* 列表 */
.list { display: flex; flex-wrap : wrap; padding : 90rpx 0 30rpx 30rpx;
background-color: #3f3f3f; }
.list .item { font-size: 24rpx; text-align: center; width : 210rpx; margin-top :
30rpx; margin-right : 30rpx; }
.list .item image{ height : 325rpx; width : 100%; box-shadow: 3px 3px 4px #111; }
.list .item text { width : 100%; height : 32rpx; margin-top : 10rpx; color :
#fff; line-height: 32rpx; display: block;white-space:nowrap;
overflow:hidden; text-overflow:ellipsis; }

```

6.4.2 详情页

详情页整体布局比较简单，分为基本信息、简介、主创三块，在视觉上我们使用阴影让模块间产生层级的效果，在基本信息中背景使用了一张模糊后的图片。要注意的是如果页面高度不够页面背景默认会显示白色，而小程序没有暴露相关接口设置页面背景颜色，所以在详情页中我们在最外层嵌套了一个view，通过逻辑层计算的高度，设置它的最小高度，这样便能通过设置这个view的背景色达到修改页面背景色的效果，整体布局结构如图6-7所示。

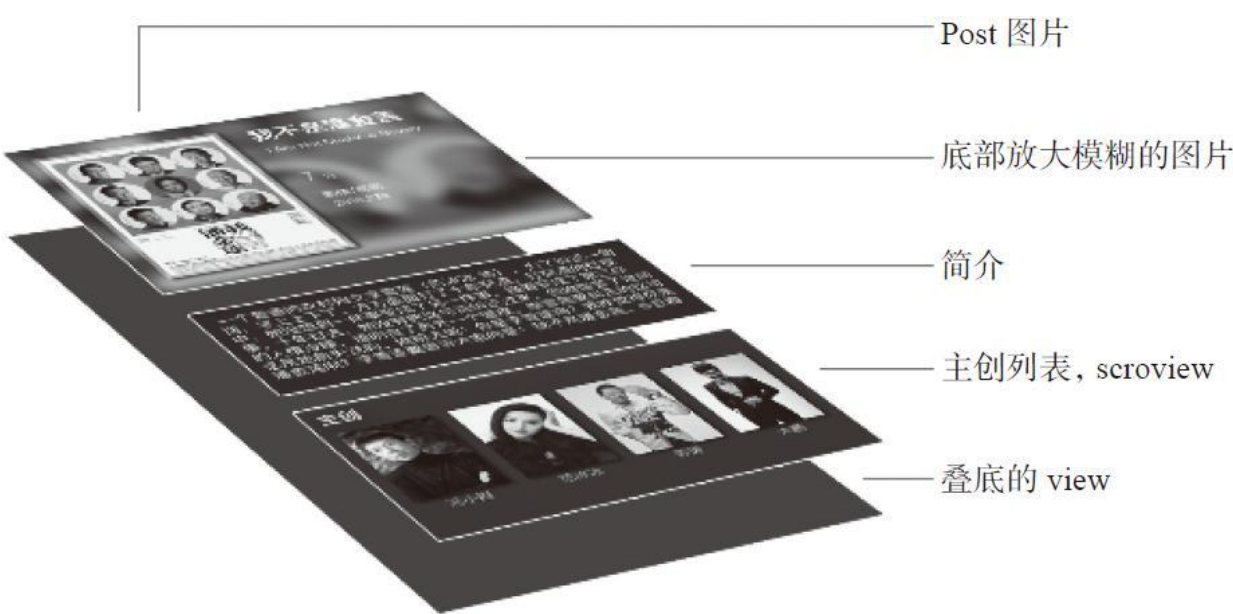


图6-7 详情页结构图

详情页布局代码如代码清单6-4和代码清单6-5所示。

代码清单6-4 detail.wxml

```
<view style="min-height : {{screen.minHeight}};background-color: #2f2f2f;">
  <view class="banner">
    <view class="poster">
      <image mode="aspectFit" src="{{movie.images.large}}"></image>
    </view>
    <view class="info">
      <view class="title">{{movie.title}}</view>
      <view>{{movie.aka[movie.aka.length - 1]}}</view>
      <view class="score">{{movie.rating.average}}<view>分</view></view>
      <view class="subinfo">
        <view>{{movie.genresStr}}</view>
        <view>{{movie.year}}上映</view>
      </view>
    </view>
    <image class="background" mode="aspectFill" src="{{movie.images.large}}">
  </image>
</view>

<view class="summary">
  {{movie.summary}}
</view>

<view class="casts">
  <view class="title">主创</view>
  <scroll-view scroll-x>
    <view class="casts-wrapper" style="width:{{movie.staff.length*183}}rpx;">
      <view class="avatar" wx:for="{{movie.staff}}">
        <image mode="aspectFill" src="{{item.avatars.large}}"></image>
        <view>{{item.name}}</view>
      </view>
    </view>
  </scroll-view>
</view>
</view>
```

代码清单6-5 detail.wxss

```
/* 电影简介 */
.banner { overflow: hidden; height : 535rpx; background-color: #fff;
  position: relative; box-shadow : 0 0 10px #111; border-bottom : solid 1px
  #000; background-color: #222; }
.banner .background { width : 130%; height : 130%; margin-left : -15%; margin-top:
  -15%; filter: blur(10px); opacity: 0.7; }

/* 海报 */
.banner .poster { box-shadow: 3px 3px 5px #222; position: absolute; left : 30rpx;
  top : 40rpx; z-index : 1; width : 300rpx; height : 435rpx; padding : 10rpx;
  background-color: rgba(255,255,255,0.4); }
.banner .poster image { width : 100%; height: 100%; }

/* 相关信息 */
.banner .info { position: absolute; color : #fff; z-index : 2; width : 320rpx;
  left : 390rpx; top : 40rpx; font-size : 24rpx; }
.banner .info .title { font-size : 46rpx; font-weight: bold; margin-bottom :
  10rpx; }
```

```
.banner .info .score { margin-top: 60rpx; font-size : 46rpx; color : #FFC125; }
.banner .info .score view{ font-size : 24rpx; display: inline; margin-left :
  10rpx; }
.banner .info .subinfo { margin-top : 20rpx; }

/* 简介 */
.summary { font-size : 28rpx; line-height : 32rpx; padding : 20rpx 20rpx 40rpx
  20rpx; color : #fafafa; background-color: #222; }

/* 演员列表 */
.casts { background-color: #2f2f2f; }
.casts .title { padding: 10rpx 20rpx; font-size : 34rpx; color : #fff;
  font-weight: bold; text-shadow : 0 2px 34px #000; background:
  -webkit-gradient(linear, 0 0, 0 100%, from(#333), to(#2f2f2f)); box-shadow :
  0 -1px 3px #111; border-top : solid 1px #444; }
.casts scroll-view { height : 310rpx; width : 730rpx; padding : 0 10rpx; overflow:
  hidden; background-color: #2f2f2f; }
.casts scroll-view .avatar { display: block; float: left; padding : 10rpx 15rpx;
  width : 153rpx; height : 222rpx; }
.casts scroll-view image { width : 153rpx; height : 222rpx; box-shadow: 3px
  3px 4px #111; }
.casts scroll-view .avatar view { text-align: center; width : 100%; font-size :
  24rpx; line-height : 26rpx; overflow: hidden; height : 52rpx; margin-top :
  10rpx; color : #ccc; }
```

6.5 页面逻辑开发

在小程序逻辑开发中，数据模型的设计特别关键，直接决定了逻辑代码和布局代码的复杂度，开发之前一定要多花时间整理数据模型。

6.5.1 首页

首页中数据模型主要分为tab和movie，在tab中我们设置了一个当前选中的index便于渲染和修改状态，movie直接赋值豆瓣返回的数据，没有进行加工。在代码设计上，我们认为列表的渲染永远和tab相关，进入首页时也只是触发选中第一个tab，这种设计思路将大大减少我们的代码量。虽然我们可以通过setData方法触发页面渲染，但我们仍然暴露了一个renderList方法，这是因为有可能在渲染页面前需要对返回的数据进行加工，这时便可在renderList方法中对数据进行处理。在编写代码过程中，一定要注意代码的可读性，尽量使用通俗易懂的编码方式。应与当前逻辑层面不相关的代码都封装到一个可理解的方法中，这样才能便于多人维护代码，首页逻辑代码如代码清单6-6所示。

代码清单6-6 home.js

```
var service = require( '../..../service/douban/douban' ),
    utils = require( '../..../common/utils/utils' ),
    _fn;

Page( {
  data : {
    movies : {},
    tabs : {
      currentIndex : 0,
      list : [{
        text : '正在热映',
        type : '1'
      }, {
        text : '即将上映',
        type : '2'
      }]
    }
  }
}
```

```

    }
  },
  onReady : function() {
    // 进入便选择第一项
    _fn.selectTab.call(this, 0 );
  },
  changeTab : function( e ) {
    var target = e.target;
    // 选中不同项
    _fn.selectTab.call( this, target.dataset.index );
  }
} );

_fn = {
  selectTab : function( index ) {
    var self = this,
        tabs = self.data.tabs;
    // 切换状态
    self.setData( {
      'tabs.currentIndex' : index
    } );
    utils.showLoading();
    // 获取数据
    service.getMovieList( tabs.list[index].type, function( data ) {
      utils.hideLoading();
      // 渲染页面
      _fn.renderList.call( self, data );
    } );
  },
  renderList : function( data ) {
    data = data || listData;
    // 可能会对数据进行处理
    this.setData( {
      movies : data
    } );
  }
}
}

```

6.5.2 详情页

详情页十分简单，仅仅是获取后台数据后进行展示，其代码如下清单6-7所示。

代码清单6-7 detail.js

```
var service = require( '.././service/douban/douban' ),
    utils = require( '.././common/utils/utils' ),
    _fn;

Page( {
  data : {
    movie : {},
    screen : {
      minHeight : 'auto'
    }
  },
  onLoad : function( query ) {
    var self = this;
    utils.showLoading();
    // 设置页面高度，避免底部出现白色区域。
    wx.getSystemInfo( {
      success : function( info ) {
        self.setData( {
          'screen.minHeight' : info.windowHeight + 'px'
        } );
      }
    } );
    // 获取数据
    service.getMovieDetail( query.id, function( data ) {
      data = data || movieDetail;
      console.log( data );
      utils.hideLoading();
      _fn.render.call( self, data );
    } );
  }
} );

_fn = {
  render : function( data ) {
    // 设置描述
    data.genresStr = data.genres.join( '/' );
    // 把演员和导演都罗列出来
    data.staff = data.directors.concat( data.cast );
    this.setData( {
      movie : data
    } );
  }
}
```


6.6 小结

豆瓣电影实例整体业务逻辑不复杂，整体实现非常简单，基于这个简单项目的代码架构，可以快速搭建出任何项目。

第7章 案例分析——驾考

目前市面上有很多与机动车驾驶人培训考试（以下简称驾考）有关的App，这些App提供了丰富的与驾考相关的服务，在市场上得到了广泛认可。本章会使用微信小程序制作一款类似的App。项目的源码地址为：<https://github.com/wxapp-book/jiakao.git>。

目前，驾考分为四个阶段，分别为科目一：理论考试；科目二：场地技能考试；科目三：道路技巧考试；科目四：安全文明驾驶考试。其中科目一与科目四为理论上机科目，科目二与科目三为实践性科目。本章练习项目的重点是开发科目一和科目四的试题练习，通过完成“顺序练习”、“模拟考试”、“成绩统计”、“错题本”等功能，使读者对小程序开发有更清晰的思路。

7.1 业务流程

开发小程序之前我们先对业务梳理并划分到不同的页面当中。这样保证不会有业务被遗漏，也有助于在开发阶段对相似的业务进行抽象，保证代码有较低的冗余度，节约开发成本。

在本章练习App中，将业务分为四个部分：首页，文科考试学习（科目一、科目四），视频教学（科目二、三学习），系统管理，下面将对各个部分的功能进行定义。

1.首页：App的主界面，提供科目一、科目二、科目三、科目四、系统管理这些功能的入口。

科目一与科目四的内容包括三种答题模式，分别为：顺序学习、模拟考试、错题本。另外，还要提供用户手动更新题库的功能。科目二与科目三需要使用一个类似轮播图的组件将学习视频展示在页面上。系统管理作为一个单独的链接放在页面上。除此之外，页面还将展示一些统计数据，如展示科目一与科目四的答题正确率和学习完成进度等，这样可以提醒用户学习的成果。

2.文科考试学习，主要提供两个页面：

1) 习题页面：用户的做题页面，该页面会被科目一和科目四的三种模式重用，分别为：模拟考试、顺序学习、错题本。以下是这三种模式的功能点：

·顺序学习：顺序学习功能会根据用户的考试科目（科目一、科目四）和车型（A1、A2、B1、B2、C1、C2）获取匹配的完整题库，然后将这些题库中的习题顺序展示给用户学习。在用户学习的过程中，完成每道题后，系统会直接对用户的答案是否正确给出结果。如果正确，自动切换至下一题，否则，标记正确答案并给出答案解析，然后将该题加入一个针对考试科目和车型的全局错题本，用户可以手动切换下一题，继续答题。同时系统会记录用户的完成进度，以供下次学习时，可以继续未完成的练习。

·模拟考试：模拟考试根据用户的考试科目和车型，从相关题库中，随机抽选正式考试要求数目的考题，作为用户的练习题。与顺序学习的共同点是：在做题过程中，会实时判断用户的作答是否正确，并将错题加入全局错题本。不同点是：模拟考试不会影响顺序学习的进度，除了会将错题加入全局错题本外，还会将其加入一个只与本次考试有关的错题本，以供用户在完成本次模拟考试后可以及时复习。

·错题本：错题本有两种，一种是全局错题本。它会把用户使用顺序学习功能时，做的错题展示给用户，这种错题本用户可以随时查看。另一种错题本是针对考试的错题本，这种错题本的进入链接只会

在每次模拟考试结束后的“考试结果”页提供，并且只会展示本次考试产生的错题。用户在完成错题本的题目后，同样会即时给出答案是否正确的判断，如果正确，则将该错题移除全局错题本。

2) 练习结果页面：练习结果页有两种状态。第一种作为模拟考试结果页，会在每次模拟考试结束后显示，主要的显示内容有本次考试的得分，成绩是否合格（科目一与科目四合格均为90分），本次考试错题本的链接等。第二种是用户完成了某个考试科目和车型对应的“顺序学习”后，看到的结果页。这个结果页会提供全局错题本的进入链接和整体的正确率。

3.科目二，科目三：这两块的业务相对简单，这里仅提供一些教学视频供在线观看。

4.系统管理：系统管理大部分的功能是与storage相关的，包括清除storage，以及向storage中保存考试车型等，这部分比较简单，所以不会作为讲解重点。

7.2 项目架构

7.2.1 功能点分析

在本项目中，科目一考试与科目四考试属于理论性考试，用户都需要一个答题页，这个答题页在两个考试科目中的功能点、用户交互，以及业务方面几乎完全相同，所以在设计页面的时候，可以考虑将其合并为一个页面，在开发的过程中尽量将差异性的东西隔离开来，保证这个页面可以满足不同科目和不同学习模式的渲染需求。

数据方面，一共调用了两个HTTP接口，这两个接口一个是根据考试科目、考试车型、学习类型（顺序学习、随机考试）获取考题的接口（以下简称考题接口），另外一个获取考试答案的映射的接口（以下简称答案映射接口）。由于驾考文科考试包含单选题与多选题，所以可能的答案组合有17种，考题接口返回的考题答案只是一个代码，这个代码代表了这17种组合的一种，所以需要一种映射机制去获取正确答案。这个接口就是这个映射的依据。

存储管理，系统中的一些常用的数据如用户的考试车型，一些需要请求，但是变化频率又不高的数据，都可以存储在微信小程序的storage中，这些数据的storage key值需要作为常量来管理。

考试页面是一个单页面系统。用户在答题的时候，系统会根据用户的点击（如用户点击上一题或下一题），以及答题正确后系统自动切换题目的方式，切换不同题目的“页面”。笔者在最初实现这个功能的时候，使用了小程序的swiper组件，因为swiper自带了左右滑动功能，可以节省页面切换的代码，但是，当用swiper去渲染“顺序学习”的页面时，题目会超过1000道，一次渲染节点数量也非常多，导致页面的性能很低，以至于无法正常调试。于是考虑将题目分批次渲染，即每次渲染10道题，当用户越过每批次的第1题或者第10题时，通过swiper的bindchange事件触发向前渲染10道题或者向后渲染10道题，但是却发现swiper组件在配置autoplay为false的情况下，当到头或者到尾的时候不会触发bingchang事件，进过反复尝试，最后放弃这个实现方案，改用了另外一种通用的单页面的解决方案，这个也是本项目的难点之一。

数据统计，用户在学习或者考试完成后，需要对考试和学习的正确率进行统计和展示，这要求系统在答题过程中，将答题的正确与否随时记录并且组成一个集合，在答题完成后，对这个集合的正确和错误的答题进行统计。

7.2.2 项目结构图

图7-1是项目结构图，下边将介绍各个部分所承担的功能。

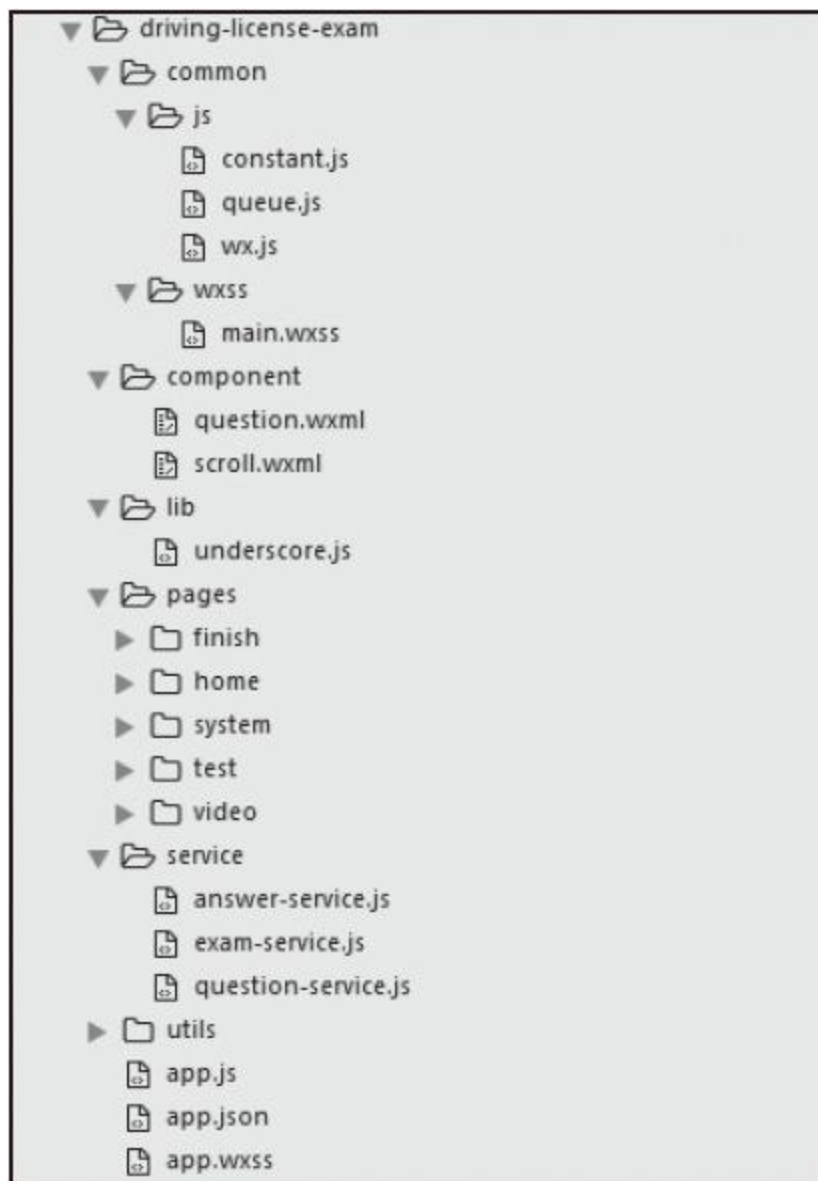


图7-1 项目结构图

common: 存放一些公共业务相关的代码。

1) **wx.js**与**constatn.js**是主要的两个文件。**wx.js**的主要功能是封装一些微信小程序的底层API。在开发过程中，有一些底层的微信小程序API是需要我们根据自身业务进行进一步封装的，例如**wx.request**这个方法，我们在请求数据的时候，需要携带聚合数据（聚合数据会在7.2.3节介绍）的**AppKey**作为一个参数，但是在多人合作开发的情况下，业务开发者不需要关心这个参数，所以，为了减少开发时间，类似这样的功能可以作为一个公共的服务提供出来。

2) **constant.js**的主要功能是保存系统的常量，例如一些常用的**storage key**、数据接口的**URL**等。

component: 存放的是一些公用的小程序模板（**template**）。

lib: 存放第三方的类库。这里介绍下**underscore.js**。微信小程序的开发是基于数据模型的，所以，会有大量的数据操作，比如列表查询、数据格式转换等工作。在这里选择**underscore.js**作为数据处理的工具，它有很多优势，如十分小巧，压缩后只有**4KB**；使用方法简单，它提供了几十种函数式编程的方法，大大方便了**JavaScript**的编程；覆盖面广，包含了操作集合、数组、函数和对象的方法；社区人气高，不用担心技术支持的问题。

pages: 存放页面代码。

1) **finish**: 科目一和科目四练习结束的页面。

2) **home**: 首页。

3) **system**: 系统配置页面。

4) **test**: 科目一和科目四的考题学习页。

5) **video**: 科目二和科目三播放视频的页面。

service: 存放公用服务代码。

1) **exam-service.js**: 考题接口每次请求的数据量都比较大，但是考题数据的变化频率却不高，所以如果是请求顺序学习的数据（全量题库），可以将数据存储在微信小程序的**storage**中，这样可以减少用户流量的开销，也可以大大减少读取数据的等待时间。因此，可以抽象出一个服务层专门提供操作题库的功能，如获取题库数据，将题库数据从**storage**读取/写入等功能以供统计缓存管理等功能使用。

2) **answer-service.js**: 答案映射接口与考题接口也比较相似，变化频率也不是很高，但是返回的数据不方便操作（见图7-4），需要将其转化为可操作的数据格式。所以，针对这个接口，也可以提炼一个服务。这个服务包含的功能有：获取答案映射数据，将答题映射从**storage**读取/写入等功能。

3) `question-service.js`: 系统在渲染考题和判断用户作答是否正确的时候，会用到一些重用率很高的功能，比如判断当前考题是单选题还是多选题的方法，还有对比用户答案与正确答案的方法等，这些与操作考题操作有关的方法，统一由这个服务来提供。

7.2.3 数据接口

本章一共调用了两个数据接口，分别为考题接口和答案映射接口，这两个接口的数据提供者并非作者本人，而是调用了第三方数据供应商聚合数据的API。

聚合数据是国内领先的基础数据服务商，B2D开发者服务的先行者。平台是以自有数据为基础，各种便捷服务整合以及第三方数据接入，为互联网开发全行业提供标准化API技术支撑服务的DaaS平台。



图7-2 聚合数据首页

聚合数据的官网是<https://www.juhe.cn/>，首页如图7-2所示，开发者可以从中获取各种实用的数据接口，比如IP查询、万年历等。这些接口有免费接口也有付费接口。同时聚合数据会不定期地组织一些活动，比如本章使用的这两个接口就是在活动期间被聚合数据免费开放出来的。

本章调用的两个接口都需要传递由聚合数据分配给开发者的Appkey。获取Appkey需要注册聚合数据的会员以及通过审核，由于篇幅的关系就不详细描述这个过程了。读者在得到Appkey之后，需要按照官网接口文档的要求将Appkey加入请求中，才能获取到正确的数据。

图7-3与图7-4分别为试题接口与答案映射接口的数据返回样例，读者可以先对这两个接口的数据格式做个简单了解。下一节将详细介绍驾考App的代码逻辑和实现细节。

```
{
  "error_code": 0,
  "reason": "ok",
  "result": [
    {
      "id": 12,
      "question": "这个标志是何含义？", //问题
      "answer": "4", //答案
      "item1": "前方40米减速", //选项，当内容为空时表示判断题正确选项
      "item2": "最低时速40公里", //选项，当内容为空时表示判断题错误选项
      "item3": "限制40吨轴重",
      "item4": "限制最高时速40公里",
      "explains": "限制最高时速40公里：表示该标志至前方限制速度标志的路段内，机动车行驶速度不得超过标志所示数",
      "url": "http://images.juheapi.com/jztk/c1c2subject1/12.jpg" //图片url
    }
  ]
}
```

图7-3 试题接口返回数据

```
{
  "error_code": 0,
  "reason": "success",
  "result": {
    "1": "A或者正确",
    "2": "B或者错误",
    "3": "C",
    "4": "D",
    "7": "AB",
    "8": "AC",
    "9": "AD",
    "10": "BC",
    "11": "BD",
    "12": "CD",
    "13": "ABC",
    "14": "ABD",
    "15": "ACD",
    "16": "BCD",
    "17": "ABCD"
  }
}
```

图7-4 答案映射接口返回数据

7.3 代码分析

7.3.1 小程序底层代码封装

小程序提供了一个`wx.request`方法，开发者通过它，以发送HTTPS请求的方式获取外部数据。但是它限制了并发数为5个，超出后将会报错并且阻止超出的请求发送。代码清单7-1对该方法进行了封装，其目的是：第一将聚合数据的Appkey封装在其中；第二通过调用`wx.showToast`方法，保证在请求过程中阻塞用户的操作；第三确保如果并发量高于5个的时候，不会有请求被阻止或者失败。

代码清单7-1 `wx.request`（）封装

```
callNum : 0, //全局变量，表示当前队列中的异步请求数量，不得超过5个
queryJH : function(args){
    wx.showToast({
        title:"数据加载中",
        icon:"loading",
        duration: 10000
    });
    var ajaxQueue = queue.getQueue("wxAjaxQueue");//初始化请求队列
    var fire = function(){//在每次请求结束后被调用
        if(handle.callNum<5&&ajaxQueue.callbackArray.length>0){
            ajaxQueue.fire();//触发队列中下个ajax请求
            handle.callNum = handle.callNum+1;
        }
    };
    var query = function(){//请求方法体
        var AppKey = constant.AppKey;
        var url = args.url;
        var data = args.data||{};
        var complete = args.complete;
        var fail = args.fail;
        data.key = AppKey;//聚合数据的Appkey

        wx.request({
            url : url,
```

```

        data: data,
        complete:function(resp){
            fire();
            if(typeof complete === "function"){
                complete(resp);
            }
            if/ajaxQueue.getSize()<=0){
                wx.hideToast();
            }
            handle.callNum--;
        }
    });
};
if/ajaxQueue && ajaxQueue.callbackArray.length>=0){//程序入口,
    ajaxQueue.add(args,query);//在队列中将请求的方法体和请求参注册, 并等待调用
    fire();
}
},

```

代码清单7-2是代码清单7-1中queue对象的源码，它是确保并发超过5个时，没有请求被阻止的关键。这个对象的功能类似Jquery的callback对象。它通过原型链对外提供了两个方法，一个是queue.add（），这个方法是用来将请求的参数与回调函数保存到一个队列中。另外一个方法是queue.fire（），这个方法取出队列中的第一个方法和参数，执行并将其移除。

代码清单7-2 queue.js

```

/*
 *common/js/queue.js
 *队列对象
 */
var us = require('../lib/underscore.js');
var queue = function(name){//构造方法
    this.name = name;
    this.callbackArray = [];
};
queue.prototype = {//原型方法
    getName:function(){
        return this.name;
    },
    add:function(){
        this.callbackArray.push({
            callback:us.last(arguments),//传递时, 最后一个参数为函数调用体
            args:arguments//函数传入参数
        });
    },
}

```

```
    fire:function(){//
        var fireObj = this.callbackArray.shift();//获取当前队列中的第一个对象
        fireObj.callback(fireObj.args);//执行
    },
    getSize:function(){
        return this.callbackArray.length;
    },
};

var queues = {
    wxAjaxQueue:new queue("wxAjaxQueue")
};
var handle = {
    getQueue:function(name){
        if(queues[name]){
            return queues[name];
        }
    }
};
module.exports = handle;首页
```

7.3.2 首页

图7-5为首页的效果图。首页的主要功能是提供各种驾考科目的学习的入口以及系统配置入口。上文已经介绍过，科目一与科目四的考题练习页的业务与用户交互十分相似，所以使用了相同的页面，通过给该页面传递不同考试科目、考试车型还有学习类型，来加载不同的数据。

代码清单7-3是首页跳转科目一科目四的学习页的渲染代码，由于代码相似，所以只展示了科目一的代码。顺序学习、模拟考试、错题本都是通过goTest方法完成的，而且它们的节点上都有属性data-testType和data-subject。data-testType用来保存学习类型，data-subject用来保存考试科目。当用户点击这些链接后，后台会将这两个参数传递到pages/test/test页面。不过在点击“错题本”时除了这两个参数以外，还会把storage中的错题记录读取出来传递。

WeChat

首页



1 科目1 完成情况1.0% 正确率30.3%

☰ 顺序学习

🔄 模拟考试

📄 错题本

↓ 更新题库

2 科目2



3 科目3



4 科目4 完成情况0.4% 正确率80.0%

☰ 顺序学习

🔄 模拟考试

📄 错题本

↓ 更新题库

图7-5 首页

代码清单7-3 科目一科目四渲染逻辑

```
<view class="subject subject1 e1">
  <view class="logo-subject"></view>
  <view class="subject-title e1">
    科目1
    <block wx:if="{{e1Record}}">
      <view class="recordMsg">完成情况{{e1Record.complete}} 正确率
{{e1Record.correct}} </view>
    </block>
  </view>
  <view class="selection">
    <view bindtap="goTest" data-testType="order" data-subject="1"
class="e_selection s1">
      <view class="inner"><view class="logo"></view>
      顺序学习
    </view>
    </view>
    <view bindtap="goTest" data-testType="rand" data-subject="1"
class="e_selection s2">
      <view class="inner"><view class="logo"></view>
      模拟考试
    </view>
    </view>
    <view bindtap="goTest" data-testType="error" data-subject="1"
class="e_selection s3">
      <view class="inner"><view class="logo"></view>
      错题本
    </view>
    </view>
    <view bindtap="updateExam" data-subject="1" class="e_selection s4">
      <view class="inner"><view class="logo"></view>
      更新题库
    </view>
    </view>
  </view>
</view>
```

7.3.3 答题页

图7-6为考试页面的效果图，从图中我们可以看到该页面的包含的数据有：考题、考试科目、考试车型、考试类型、当前的完成进度等。包含的操作有：选择答案、答题、翻页、结束考试等。这个页面的数据模型为图7-7，从这张图中我们可以找到大部分页面所需的数据，下边对一些特殊的字段做一些解释。

返回

WeChat

c1 科目1 顺序学习

结束



1. 这个标志是何含义?

- A. 小型车车道
- B. 小型车专用车道
- C. 多乘员车辆专用车道
- D. 机动车车道

确定

上一页

1/1229

下一页

图7-6 答题页

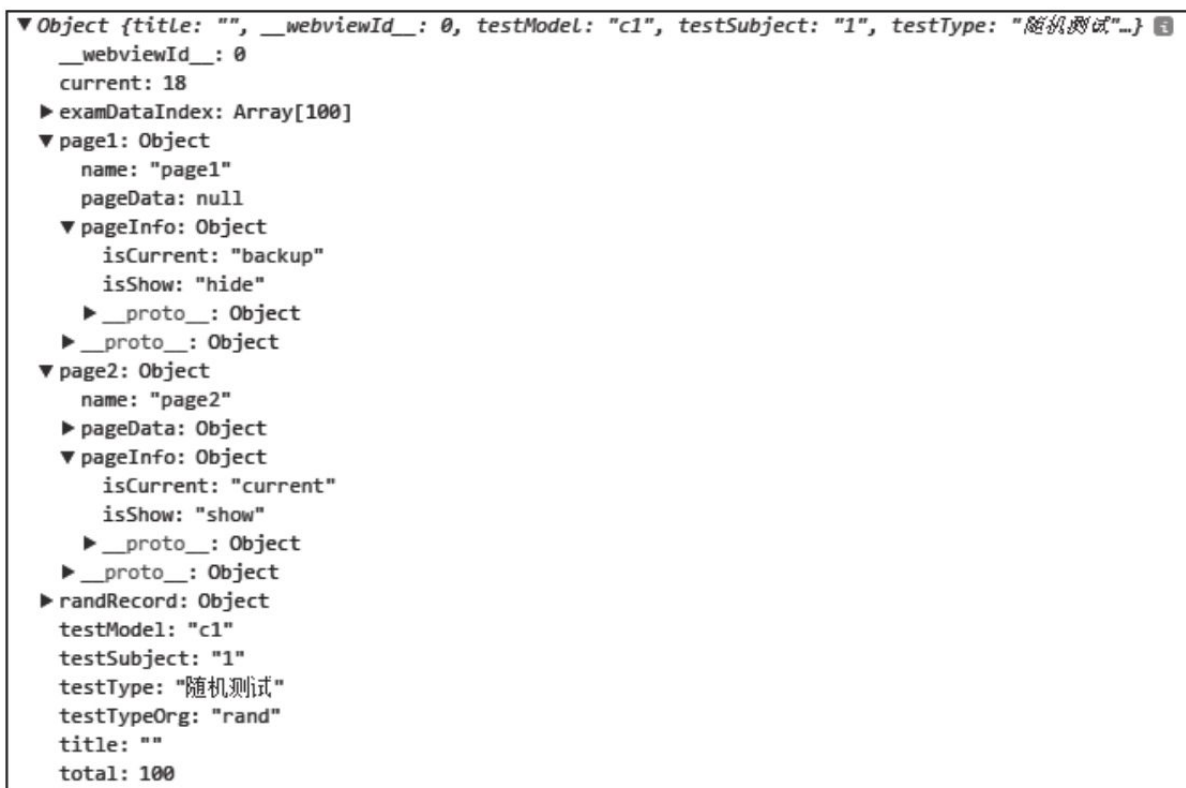


图7-7 答题页数据模型

examDataIndex: 本次学习所需要的全量考题数据。比如用户选择了C1车型科目一的顺序学习，这个字段里边就存储科目1车型C1的全量题库。如果用户选择了随机考试，系统会调用考题接口，获取科目一车型C1题库中的随机100道题保存在这个字段中。如果用户选择了错题本，则考题是通过参数传递的，这个参数是一个由考题Id（考题接口返回的每个考题都有一个Id）组成的数组。得到这个Id数组后，将其映射成为考题数组并保存在其中。

page1/page2: 存储当前页面的考题数据，这两个字段会交替保存该数据，即当一个作为存储字段时，另外一个会被置空。这样设计的原因是答题页是一个单页面，所有的题目必须在同一个页面展示，但是不能一次渲染完，因为节点过多性能很差，因此一次只会渲染一道题，而且还要考虑页面切换的效果。所以在开发渲染方法时，设计了两个渲染view，这两个view会交替展示考题。

代码清单7-4是答题页渲染的主要代码，两个渲染的view（data-block-name="1"与data-block-name="2"）分别由page1字段和page2字段渲染。这段代码只是渲染逻辑的框架，考题的数据是由./component/question-template.wxml的模板生成，这个模板的渲染只需要把examDataIndex列表中的考题数据传入即可。这段代码需要关注的是渲染view的两个动态class，分别由pageInfo.isCurrent和pageInfo.isShow的值来展示。pageInfo.isCurrent是用来表示这个view是否为当前用户正在操作的界面，可能的值有current、backup，这个class并不会控制页面的样式变化，只是用来标识该view是否作为展示view。pageInfo.isShow是用来控制页面的显示和隐藏的，可能的值有show、hide，这个class不会影响业务逻辑。

代码清单7-4 test.wxml

```
<view class="dle-body">
  <view class="test">
    <view data-block-name="1" class="{{page1.pageInfo.isCurrent}}
      {{page1.pageInfo.isShow}} page">
      <template is="question-template" data="{{...page1}}"/>
    </view>
  </view>
</view>
```

```
</view>
<view data-block-name="2" class="{{page2.pageInfo.isCurrent}}"
  {{page2.pageInfo.isShow}} page">
  <template is="question-template" data="{{...page2}}"/>
</view>
</view>
</view>
```

代码清单7-5是用来获取答题页显示层和隐藏层数据的方法，在开发中经常会遇到需要知道哪个view是显示view的场景，如切题时判断哪个页面应该被隐藏，哪个应该被渲染。页面的变化就是数据模型的变化，所以在设计数据模型时要考虑到数据可以描述页面的所有变化，在设计界面的时候，需要覆盖这个数据模型的所有变化，而不是像开发H5时，通过JS去改变DOM。

代码清单7-5 获取页面显示层和隐藏层的数据

```
getPageData:function(){
  var page1 = us.last(getCurrentPages()).data.page1;
  var page2 = us.last(getCurrentPages()).data.page2;//得到渲染的两个数据对象
  var showPage={},hidePage={};
  if(page1.pageInfo.isCurrent==="current"){//page1为显示数据
    showPage = {
      name:"page1",
      data:page1
    };
    hidePage = {
      name:"page2",
      data:page2
    };
  }else if(page2.pageInfo.isCurrent==="current"){//page2为显示数据
    showPage = {
      name:"page2",
      data:page2
    };
    hidePage = {
      name:"page1",
      data:page1
    };
  }
  return {调用者可以通过这个方法获取当前显示层和隐藏层的数据
    showPage:showPage,
    hidePage:hidePage
  };
}
```

代码清单7-6是用来根据页码值获取考题的代码。由于系统一次只会渲染一道题在页面上，所以会经常有题目查询的需求。不论用户选择的答题模式是顺序学习、模拟考试还是错题本，保存在 `examDataIndex` 字段中的考题列表的数据格式是相同，页码的值本质就是当前考题在考题列表中的编号，所以本方法就是从这个数组中根据编号，获取考题数据。

代码清单7-6 根据页面标号获取考题

```
getQuestionByPageNum:function(nextPageNum){
    var examData = _fn.getExamIndex();//从数据模型中得到渲染题列表
    if(nextPageNum-1>=0 && nextPageNum-1<=examData.length){
        var question = examData[nextPageNum-1];
        question.index = nextPageNum;
        return {
            question:question,
            total:examData.length,
            current:nextPageNum
        };
    }else{
        return false;
    }
},
```

下边将介绍考题渲染的业务逻辑，图7-8是渲染的数据，其中大部分为考题数据，还有一些数据是在用户的操作过程中产生的。

·**answer**: 本题的答案编号，系统会用这个编号与答案映射接口返回的数据做匹配，从中找到该题对应的答案选项，图7-8中表明该题的答案为D。

·**correctAnswerMap**: 答案的数据化, 这个数据最终会渲染到相应的答案上, 用作高亮正确答案和标记用户是否答题正确。

·**judgRes**: 答题是否正确的判断。

·**lockQuestion**: 用户对该题的操作锁, 在用户提交答案后, 这个值就会变为**true**, 表示用户不可再答题。

·**showExplains**: 是否显示正确答案以及解释, 在用户答题错误后, 这个值会变为**true**。

·**userSelect**: 用户的选项数据化, 这个数据最终会渲染到相应的答案上, 用作高亮用户当前的选择。

代码清单7-7是考题渲染的逻辑。考题页面可以分为三个状态, 分别为页面初始化、用户选中答案、用户提交答案三个阶段。初始化状态就是用户在页面上未做任何操作, 只展示考题和考题选项, 此时的渲染数据只有题目和答案选项。考题渲染完成后, 用户可以点击选项使页面发生变化, 此时的渲染数据添加了**userSelect**对象。此处需要注意, 虽然小程序模板的渲染数据需要父页面传入, 但事件是可以与父页面共用一个**page**对象的。第三个状态在用户确定答案后才会触发。此时会生成**correctAnswerMap**、**judgRes**、**lockQuestion**、**showExplains**对象对页面进行渲染。


```

▼ page1: Object
  name: "page1"
  ▼ pageData: Object
    answer: "4"
    ▼ correctAnswerMap: Object
      item1: "in-correct"
      item2: "in-correct"
      item3: "in-correct"
      item4: "correct"
      ► __proto__: Object
    explains: "此为机动车车道，比多乘员车辆专用车道少俩人。"
    id: 1
    index: 1
    item1: "小型车车道"
    item2: "小型车专用车道"
    item3: "多乘员车辆专用车道"
    item4: "机动车车道"
    judgeRes: false
    lockQuestion: true
    question: "这个标志是何含义？"
    showExplains: true
    url: "http://images.juheapi.com/jztk/clc2subject1/1.jpg"
  ▼ userSelect: Object
    item1: "selected"
    ► __proto__: Object
  ► __proto__: Object

```

图7-8 页面渲染数据

代码清单7-7 考题渲染

```

<template name="question-template">
  <view class="question-view">
    <block wx:if="{{pageData.url}}">
      <view><!--渲染考题图片-->
        <image class="picture" src="{{pageData.url}}" ></image>
      </view>
    </block>
    <block wx:if="{{pageData.id}}"><!--渲染考题文案-->
      <view class="question">{{pageData.index}}. {{pageData.question}}</view>
    </block>
    <view class="answers"><!--渲染答案-->
      <block wx:if="{{pageData.item1}}"><!--选项有三种状态：未选中，已选中，判断后-->

```

```

        <view class="item-1" data-value="1" data-item-id="{{pageData.id}}"
bindtap="selectItem" class="answer-item {{pageData.userSelect.item1}}
{{pageData.correctAnswerMap.item1}}">A.{{pageData.item1}} </view>
    </block>
    <block wx:if="{{pageData.item2}}">
        <view class="item-2" data-value="2" data-item-id="{{pageData.id}}"
bindtap="selectItem" class="answer-item {{pageData.userSelect.item2}}
{{pageData.correctAnswerMap.item2}}">B.{{pageData.item2}}</view>
    </block>
    <block wx:if="{{pageData.item3}}">
        <view class="item-3" data-value="3" data-item-id="{{pageData.id}}"
bindtap="selectItem" class="answer-item {{pageData.userSelect.item3}}
{{pageData.correctAnswerMap.item3}}">C.{{pageData.item3}}</view>
    </block>
    <block wx:if="{{pageData.item4}}">
        <view class="item-4" data-value="4" data-item-id="{{pageData.id}}"
bindtap="selectItem" class="answer-item {{pageData.userSelect.item4}}
{{pageData.correctAnswerMap.item4}}">D.{{pageData.item4}}</view>
    </block>
</view>
<block wx:if="{{pageData.id}}"><!--提交答案按钮-->
<view class="do-answer" data-item-id="{{pageData.id}}" bindtap="doAnswer">确
定</view>
</block>
<block wx:if="{{pageData.showExplains===true}}"><!--答案解析，用户未提交答案，或
者答案正确，不会显示，只有答题错误才会显示-->
<view class="correct-answer">
    <view class="answer-msg"><!--答案选项-->
        正确答案：
        <block wx:if="{{pageData.answer===1}}">
            A.{{pageData.item1}}
        </block>
        <block wx:elif="{{pageData.answer===2}}">
            B.{{pageData.item2}}
        </block>
        <block wx:elif="{{pageData.answer===3}}">
            C.{{pageData.item3}}
        </block>
        <block wx:else="{{pageData.answer===4}}">
            D.{{pageData.item4}}
        </block>
    </view>
    <view class="answer-explain"><!--解释-->
        答案释义：
        {{pageData.explains}}
    </view>
</view>
</block>
</view>
</template>

```

代码清单7-8用来高亮用户选择的选项，也就是上文说的userSelect字段的生成方式。userSelect对象的key值的可选值为item1、item2、item3、item4，分别对应了页面上的A、B、C、D选项，用户每次选择选项后，都会刷新这个对象。注意，这里需要考虑考题为多选题的情

况。标记后，用户选择的选项的key值为selected，未选择的选项不会存入该对象。在这个数据创建完成后，会调用render.renderCurrentPage()来刷新这个页面。刷新后，被选中的考题选项会多出一个值为selected的class。本页面的wxss会对这个class进行高亮的样式处理。

代码清单7-8 高亮选中选项

```
highlightItem:function(questionId,selectValue){
    var curQuestion = test.getCurrentQuestion().question;//得到当前显示的考题
    var isLock = curQuestion.lockQuestion;//锁住回答
    if(isLock){
        return;
    }
    var questionType = questionService.getQuestionType(curQuestion.answer);
    //计算该题是单选还是多选
    var userSelect = curQuestion.userSelect||{};
    if(questionType===1){//如果是单选，则清空已经选的选项
        userSelect = {};
    }
    userSelect["item"+selectValue] =
        userSelect["item"+selectValue]?false:'selected';
    curQuestion.userSelect = userSelect;//构建userSelect对象
    render.renderCurrentPage();
},
```

代码清单7-9是用来判断当前用户的答题结果是否正确的代码。开始时系统会获取userSelect字段标记的选项。之后遍历这个对象中的所有key，将key转换成一个数字，item1为1，item2为2，item3为3，item4为4，然后将这些数字拼为一个字符串（注意多选题的情况），与answer字段对应的答案映射接口处理后的答案字符串（见图7-9）进行匹配，如果匹配成功，则判断当前答题正确。如图7-8数据的answer字段为4，对应答案映射接口的数据为4，表示选项D为正确。判断完成后，会返回一个与userSelect相似的对象，不同的是，会将所有选项的判断结果标记出来。数据创建完成后，会调用render.renderCurrentPage

() 方法刷新答案选项的class，正确答案的class是correct，错误为in-correct。然后本页面的wxss会对有correct和in-correct的class节点做相应的样式处理。

代码清单7-9 判断答案是否正确

```
judge:function(questionId){
    var curQuestion = test.getCurrentQuestion().question;
    var userSelect = curQuestion.userSelect || {};
    var item1Status = userSelect.item1==='selected'?"1":"";
    var item2Status = userSelect.item2==='selected'?"2":"";
    var item3Status = userSelect.item3==='selected'?"3":"";
    var item4Status = userSelect.item4==='selected'?"4":"";
    //将用户选择的选项转换为答案映射接口处理后的字符串
    var userAnswerStr = item1Status+item2Status+item3Status+item4Status;
    var correctAnswer = questionService.getQuestionAnswer(curQuestion);
    //获取数据映射接口处理后的答案数据
    curQuestion.correctAnswerMap = correctAnswer.answerObj;
    var judgeRes = false;
    //如果相同则表示正确，否则为错误
    if(userAnswerStr===correctAnswer.answerStr){
        judgeRes = true;
    }
    curQuestion.judgeRes = judgeRes;
    record.doRecord({
        questionId:questionId,
        judgeRes:judgeRes
    });
    return judgeRes;
},
```

```
{
  "1": ["1", "1"], //A或正确,
  "2": ["2", "2"], //B或错误,
  "3": ["3"], //C
  "4": ["4"], //D
  "7": ["12"], //AB
  "8": ["13"], //AC
  "9": ["14"], //AD
  "10": ["23"], //BC
  "11": ["24"], //BD
  "12": ["34"], //CD
  "13": ["123"], //ABC
  "14": ["124"], //ABD
  "15": ["134"], //ACD
  "16": ["234"], //BCD
  "17": ["1234"] //ABCD
}
```

图7-9 处理后的答案映射

代码清单7-10是用来刷新考题页数据的代码，这个方法会在上文讲到的用户选择答案以及确定答案后被调用。

代码清单7-10 刷新当前页面数据

```
renderCurrentPage:function(){
    var curQuestionObj = test.getCurrentQuestion();//获取当前的考题
    var curQuestion = curQuestionObj.question;//获取新的渲染数据
    if(!curQuestion){
        return false;
    }
    var pages = _fn.getPageData();
    var showPage = pages.showPage;//获取显示页的数据
    var showPageName = showPage.name;
    var showPageData = showPage.data;

    showPageData.pageData = curQuestion;//更新渲染页的数据

    newRenderData = {};
    newRenderData[showPageName]=showPageData;
    newRenderData.total = curQuestionObj.total;//更新页码数据
    newRenderData.current = curQuestionObj.current;
    getCurrentPages()[0].setData(newRenderData);//渲染
    return true;
},
```

代码清单7-11是用来处理用户点击下一页的代码，因为在页面切换的时候会留有0.5秒的延迟，在这0.5秒期间，是不允许用户继续做翻页操作的。所以使用变量pageLock来锁住翻页操作。这个锁会在翻页完成后解开。之后的render.renderNextPage（）方法用来将数据刷入考题隐藏页。但是这个方法并没有做页面效果的切换，只返回了一个执行结果，如果为true时，会执行render.exchangeShowHide（）方法，这个方法就是执行页面切换转换，显示隐藏考题页状态的方法，将pageLock设为false也是在这个方法中执行的。

代码清单7-11 翻页代码

```
goNext:function(){
    if(pageLock){
        return;
    }
    pageLock = true;
    if(render.renderNextPage()){
        render.exchangeShowHide();
    }
},
```

代码清单7-12是用来处理切换考题的代码。该方法首先获取当前题目的下一道题，然后将考题传递给render.exchangePageContent（）方法。这个方法的主要逻辑是将传入的考题数据，渲染到当前隐藏考题页中，将显示考题页数据清空。

代码清单7-12 切换考题

```
renderNextPage:function(){
    //从examDataIndex中获取当前题目的下一道题
    var nextQuestionObj = test.getNextQuestion();
    //将这道题渲染在页面上
    return render.exchangePageContent(nextQuestionObj);
},
exchangePageContent:function(question){
    var newQuestionObj = question;
    var newQuestion = newQuestionObj.question;
    if(!newQuestion){
        return false;
    }
    //获取显示页和隐藏页的数据
    var pages = _fn.getPageData();
    var showPage = pages.showPage;
    var hidePage = pages.hidePage;
    var showPageName = showPage.name;
    var showPageData = showPage.data;
    var hidePageName = hidePage.name;
    var hidePageData = hidePage.data;

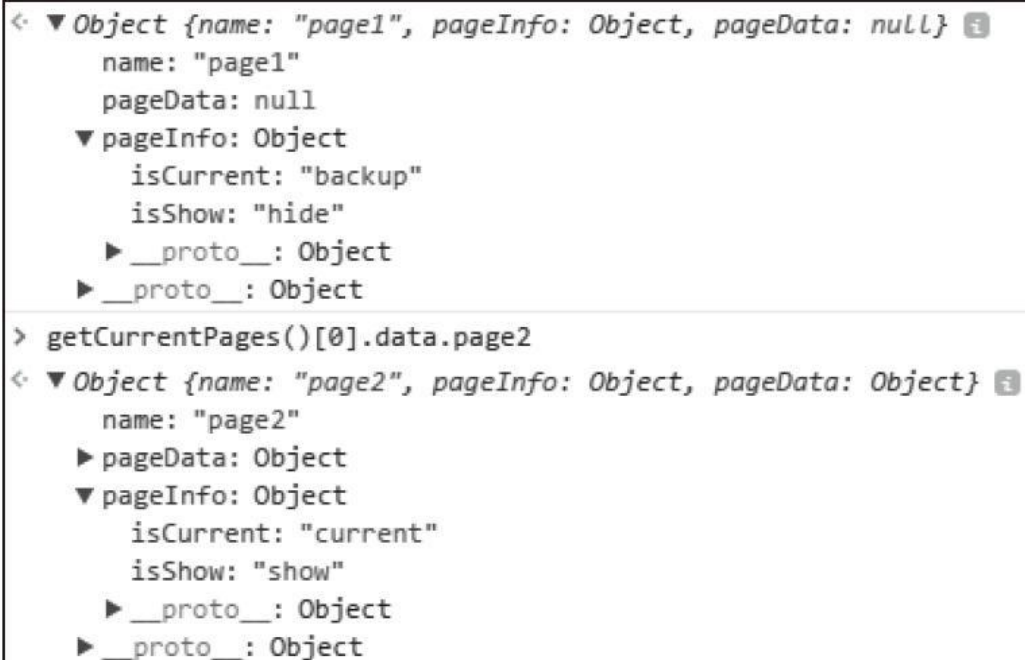
    if(!showPageData.pageData){
        //页面初始化第一次被调用，两个层都没有被渲染，就选当前的显示层
        showPageData.pageData = newQuestion;
        hidePageData.pageData = null;
    }else{
        //第一次之后的渲染，就选择当前的渲染层渲染
        hidePageData.pageData = newQuestion;
        showPageData.pageData = null;
    }
    newRenderData = {};
    newRenderData[showPageName]=showPageData;
    newRenderData[hidePageName]=hidePageData;
    newRenderData.total = newQuestionObj.total;
    newRenderData.current = newQuestionObj.current;
    getCurrentPages()[0].setData(newRenderData);
    return true;
},
```

代码清单7-13是用来实现翻页效果的代码，基本原理就是将两个page对象的pageInfo（如图7-10所示）的值互换。这里使用了500毫秒的

延迟是因为该方法在答题正确后会被重用，500毫秒会给用户一小段时间看到自己的答题结果为正确。

代码清单7-13 翻页效果

```
exchangeShowHide:function(){
    var pages = _fn.getPageData();
    var showPage = pages.showPage;
    var hidePage = pages.hidePage;
    var showPageName = showPage.name;
    var showPageData = showPage.data;
    var hidePageName = hidePage.name;
    var hidePageData = hidePage.data;
    setTimeout(function(){
        showPageData.pageInfo.isCurrent="backup";
        hidePageData.pageInfo.isCurrent="current";
        showPageData.pageInfo.isShow="hide";
        hidePageData.pageInfo.isShow="show";
        newRenderData = {};
        newRenderData[showPageName]=showPageData;
        newRenderData[hidePageName]=hidePageData;
        getCurrentPages()[0].setData(newRenderData);
        //解开考题锁
        pageLock = false;
    },500);
}
```



```
< ▼ Object {name: "page1", pageInfo: Object, pageData: null} ⓘ
  name: "page1"
  pageData: null
  ▼ pageInfo: Object
    isCurrent: "backup"
    isShow: "hide"
    ► __proto__: Object
    ► __proto__: Object
> getCurrentPages()[0].data.page2
< ▼ Object {name: "page2", pageInfo: Object, pageData: Object} ⓘ
  name: "page2"
  ► pageData: Object
  ▼ pageInfo: Object
    isCurrent: "current"
    isShow: "show"
    ► __proto__: Object
    ► __proto__: Object
```


图7-10 pageInfo对象

代码清单7-14是用来实现统计数据的代码。用户在每次答题完成后，都会将本题的结果保存到一个统计对象中（见图7-11），以供系统对用户的模拟考试或者顺序学习的成果进行评估。关于这个统计对象，顺序学习与随机考试有共同的部分，都会将用户完成的正确的题目编号与错误的题目编号添加到统计对象中。不同的是顺序学习会将用户的完成进度也保存在storage中，随机考试则会在页面数据模型中，另外保存一个针对本次考试的统计对象，同样也只有正确题目编号和错误题目编号，这个统计对象不会保存到storage中。

```
▼ randRecord: Object
  ► correct: Array[19]
  ▼ inCorrect: Array[30]
    0: 21
    1: 41
    2: 59
    3: 62
    4: 75
    5: 84
    6: 98
    7: 112
    8: 124
    9: 214
    10: 230
    11: 241
    12: 339
    13: 348
    14: 366
    15: 367
    16: 393
    17: 395
    18: 397
    19: 406
    20: 417
    21: 420
    22: 436
    23: 443
    24: 444
    25: 457
    26: 474
    27: 515
    28: 526
    29: 565
    length: 30
    ► __proto__: Array[0]
  ► __proto__: Object
```

图7-11 答题统计的数据模型

代码清单7-14 添加统计数据

```
doRecord:function(args){
    var judgeRes = args.judgeRes;
    var testType = getCurrentPages()[0].data.testTypeOrg;
    var currentIndex = getCurrentPages()[0].data.current;
    //从storage中初始化统计对象
    var orderRecord = examService.readExamRecord({
        model:getCurrentPages()[0].data.testModel,
        subject:getCurrentPages()[0].data.testSubject
    });
    if(!orderRecord){
        orderRecord = {
            correct:[],
            inCorrect:[],
            lastIndex:0
        };
    }
    var orderCorrectArr = orderRecord.correct;//获取正确序列
    var orderInCorrectArr = orderRecord.inCorrect;//获取错误序列
    if(orderCorrectArr.indexOf(currentIndex)>=0){
        orderCorrectArr[orderCorrectArr.indexOf(currentIndex)]="";
        //清空本题在正确序列的记录
    }
    if(orderInCorrectArr.indexOf(currentIndex)>=0){
        orderInCorrectArr[orderInCorrectArr.indexOf(currentIndex)]="";
        //清空本题在错误序列的记录
    }
    if(judgeRes===true){//正确添加正确序列
        orderCorrectArr.push(currentIndex);
    }else{//错误添加错误序列
        orderInCorrectArr.push(currentIndex);
    }
    if(testType==="rand"){//答题模式为随机考试
        //初始化统计对象
        var randRecord = getCurrentPages()[0].data.randRecord||{correct:[],inCorrect:
    []};
        var correctArr = randRecord.correct;
        var inCorrectArr = randRecord.inCorrect;
        if(judgeRes===true){
            //添加到正确序列中
            correctArr.push(currentIndex);
        }else{
            //添加到错误序列中
            inCorrectArr.push(currentIndex);
        }
        getCurrentPages()[0].data.randRecord = randRecord;
    }else{//答题模式为顺序学习
        var orderLastIndex = orderRecord.lastIndex;//获取答题进度
        //更新进度数据
        orderRecord.lastIndex = orderLastIndex<currentIndex?
    currentIndex:orderLastIndex;
    }
    //将数据保存到storage中
    examService.saveExamRecord({
        model:getCurrentPages()[0].data.testModel,
        subject:getCurrentPages()[0].data.testSubject,
        value:orderRecord
    });
}
```

```
} });
```

7.3.4 答题结果页

图7-12是学习结束页，用户在顺序学习或者模拟考试的时候，可以通过点击结束，查看当前类型学习的成绩。

代码清单7-15是用来渲染随机考试结束页的代码。这里需要注意的是`randRecord`这个变量，它是在答题页通过参数传递过来的，值就是答题页数据模型中的`randRecord`变量（见图7-11）。页面会根据学习类型以及正确率给用户一个总结语。代码清单7-16是用来渲染顺序学习的结束页的代码，与随机考试不同的是，统计数据对象的命名变为`orderRecord`，而且来源也不是通过页面参数，而是从`storage`中获取。

C1

科目1
顺序学习



老司机

您答对了128道题中的120道
正确率为98%

回首页

查看错题

图7-12 学习结束页

代码清单7-15 随机考试结束页渲染

```
renderRandTest:function(param){
    var randRecord = param.randRecord;
    var correctNum = randRecord.correct.length;//回答正确的题目数量
    var total = param.testType=="4"?50:100;//科目4有50道,科目1有100道
    var percentage = (correctNum * 100 / total).toFixed(2)+'%';//正确率
    var renderData = {
        model : param.model,
        subject : param.subject,
        testType: param.testType=="rand"? "随机测试": "顺序学习",
        title:_fn.getTitle(testType,percentage),//给用户一个总结语
        total:total,
        percentage:percentage,
        correct:correctNum,
        record:randRecord
    };
    us.last(getCurrentPages()).setData(renderData);
}
```

代码清单7-16 顺序学习结束页渲染

```
renderOrderTest:function(param){
    examService.readExamRecord({
        model:param.model,
        subject:param.subject,
    },function(data){
        var orderRecord = data.data;
        var correctNum = us.compact(orderRecord.correct).length;
        var total = orderRecord.total;
        var percentage = (correctNum * 100 / total).toFixed(2)+'%';

        var renderData = {
            model : param.model,
            subject : param.subject,
            testType: param.testType=="rand"? "随机测试": "顺序学习",
            title:_fn.getTitle(),
            total:total,
            percentage:percentage,
            correct:correctNum,
            record:orderRecord
        };
        us.last(getCurrentPages()).setData(renderData);
    });
}
```

7.4 小结

本章实例项目的难点有两个：

1) 单页面的设计与开发。通过交替使用两个考题渲染字段实现了一个单页面应用，从而实现了数据量很大的页面达到按需加载的目的。

2) 缓存的应用场景。将使用频率很高并且数据量很高的数据保存在缓存中，减少用户的加载等待时间以及流量的消耗，同时也提高了系统的稳定性。

第8章 案例分析——打赏

微信支付是集成在微信客户端的支付功能，用户可以通过手机快速完成支付流程。目前微信支付在实体行业和互联网行业的在线支付领域占有很大的市场份额，用户可以通过扫码支付、App支付、公众号支付等途径方便地与商家交易。现在微信的传播途径又添加了微信小程序这一利器，相信会进一步增加市场对微信支付的需求。本章通过一个简单的打赏功能来展示微信支付是如何在微信小程序中开发的。

本章的实例App不会有太多小程序在用户交互方面的功能展示，它只完成了一个主要功能，就是微信支付。用户在进入App之后，可以通过小程序的登录接口和获取用户信息接口得到自己的基本信息，如昵称、头像、所在省市等，并将其展示在页面上，随后，用户可以点击页面上的打赏按钮，触发客户端与服务器交互后，得到一次微信支付的机会，用户在完成支付后，再次进入App，会看到自己支付过的记录。

为了使读者可以更轻松地了解后台是如何与微信服务器进行交互的，本章的服务器语言选择了Nodejs进行开发，以减少用户的学习成本。

8.1 登录

8.1.1 登录流程

在讲解登录代码之前，需要先讲解一下什么是登录态和微信小程序登录态（以下简称微信登录态）。登录态就是你自己的服务器，认可当前用户是在自己的用户系统中存在，而颁发给客户端的一个凭证，客户端可以带着这个凭证，与服务器交互。服务器拿到这个登录态后可以找到自己用户系统的对应用户，从而系统可以给用户提供某些服务。微信登录态就是由微信颁发给当前用户的一个凭证，客户端可以带着这个凭证与微信交互，获取用户在微信中注册的信息。

由于微信拥有一套十分完善的用户系统，所以很多互联网公司都采用了将微信账号与自己的账号系统相关联的做法。这样既可以减少用户注册和登录的麻烦，也可以增加自身流量的来源。

微信登录开发大致可以看成两个部分：前端开发和后端开发。

前端开发主要是在每次发送请求前检查用户当前的`storage`中，是否保存了登录态，如果没有保存，则需要调用小程序的接口，获取微信登录的`code`（后边会介绍`code`的用途），然后根据`code`向服务器请求登录态。客户端拿到登录态后，每次向自己服务器发送请求的时候，

需要携带登录态一起发送到服务器端，如果服务器端判断当前的登录态不合法或者失效，则需客户端重新走微信登录程序，并且向自己的服务器获取登录态。

后端开发的关注点在于判断用户的每次请求中是否携带本系统的登录态，如果有登录态，则验证该登录态是否正确。如果没有登录态，则需要判断用户是否已经在微信登录，如果已完成登录，则使用微信登录code匹配本系统的用户，并向客户端发送登录态。如果未登录，则需要提示客户端登录。

用户每次微信登录成功之后，通过自己的服务器向微信服务器获取用户的微信登录态，包含session_key和openId。请注意这两个数据在开发中会经常用到，如果泄露将涉及安全问题，所以一定要在服务器端获得并保存。获取完成后，服务器应当将这些信息与自己的用户系统进行关联，关联成功后，需要生成一个基于自己系统的登录态，这个登录态需要考虑一些安全因素，首先要保证这个登录态的随机性，还要保证这个登录态要有一定的时效性，最后将这个登录态返回给客户端。图8-1是微信官方推荐的微信登录流程图。

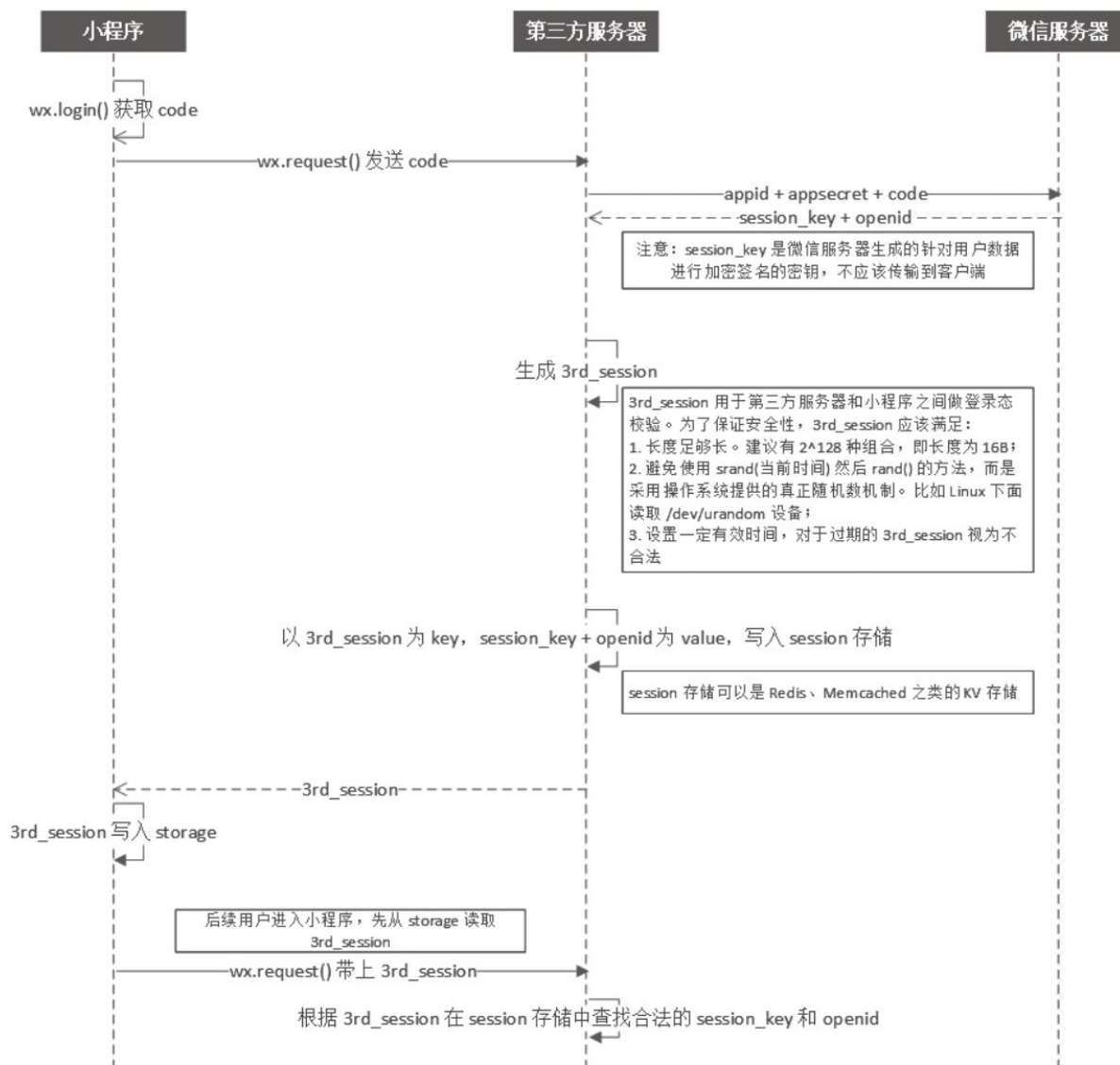


图8-1 微信登录流程

8.1.2 源码讲解

图8-2是打赏系统的主页。页面的逻辑比较简单，用户进入页面后，客户端自动请求后台接口，获取用户信息。如果后台没有该用户的信息，则客户端走微信登录，获取保存在微信服务器中的用户信息，然后将此信息传给后台。后台保存数据后，将本用户信息和登录态返回客户端，客户端将该登录态保存到本地的storage中，供其他请求使用。



图8-2 打赏页面

代码清单8-1是index页面的onload方法。这段代码的功能是在进入后，先获取用户信息然后将用户信息渲染到页面上，`user.getUserInfo`方法用于获取用户信息。

代码清单8-1 打赏页入口

```
onLoad: function () {
  console.log('onLoad');
  user.getUserInfo(function(userInfo){//得到用户的信息
    console.log(userInfo);
    _fn.render({userInfo:userInfo});//将用户信息展示在页面上
  });
}
```

代码清单8-2是`user.getUserInfo`的代码。它接受一个回调方法作为参数，并在成功得到用户信息后执行。在这段代码中，`user.wxUserRequest`是一个陌生的方法，它的本质是包装了`wx.request`这个小程序原生API。包装这个API的原因是服务器提供的一些请求，需要客户端提供存储在`storage`中的登录态，登录态的验证与生成涉及前后台比较复杂的交互，比如客户端在何时进行微信登录，何时重新获取登录态等。为了方便这些逻辑的重用，需要将验证和生成登录态的逻辑与发送请求的逻辑抽象在一个方法中。

代码清单8-2 `getUserInfo`代码

```
getUserInfo:function(callback){
  var url = constant.host + constant.path.getUserInfo;//请求地址
  user.wxUserRequest({
    url:url
  })
}
```

```
    },function(data){
      if(typeof callBack === 'function'){
        callBack(data);
      }
    });
  }
}
```

代码清单8-3是user.wxUserRequest的代码，它接受两个参数，option和callBack。option包含的信息有请求的Url和请求需要传递的参数。callBack是请求成功后的回调函数。方法开始时，通过weixin.getSessionId（下文介绍）获取用户的登录态，这个方法会保证返回一个用户的登录态。得到登录态之后，它将与用户传递的请求参数合并，发送到客户端。当请求返回后，如果没有网络问题的话，则开始检查服务器的错误码，这个方法只检查错误码为0001的情况，在这种情况下说明服务器端认为客户端传递的登录态不合法，这时客户端需要清空storage中的这个登录态，递归这个方法。再次执行时，weixin.getSessionId方法会重新向服务器请求新的登录态，保证第二次发送请求时，服务器会认为该登录态合法，从而返回非0001的错误码，之后便可执行参数中的回调函数。

代码清单8-3 wxUserRequest代码

```
wxUserRequest:function(option,callBack){//封装微信请求
  weixin.getSessionId(function(sessionId){//这个方法会确保返回一个登录态
    if(!sessionId){
      wx.showModal({
        title:'错误',
        contetn:'获取用户登录态失败',
        showCancel:false
      });
      return;
    }
    option.data = option.data || {};
    option.data.sessionId = sessionId;//将sessionId与请求参数合并
    wx.request({
      url:option.url,
```

```

        data:option.data,
        complete:function(result){
            var resData = _fn.checkAjaxRes(result);//判断请求是否成功
            if(!resData){
                return;
            }
            if(resData.code==='0001'){//服务器认为登录态不合法
                wxService.setStorage({//清空登录态后重新执行。
                    key:"sessionId",
                    data:null,
                    complete:function(){
                        utils.wxUserRequest(option,callback);
                    }
                });
            }else{
                if(typeof callback === 'function'){
                    callback(resData);
                }
            }
        }
    });
});
});//end
}

```

代码清单8-4是weixin.getSessionId的代码。这个方法的功能集获取与返回登录态于一身。它只接受一个回调函数作为参数，这个回调函数只有在成功从storage中取得登录态后会被调用。但是storage中可能会没有这个数据，这时，就需要客户端向服务器端发送请求（该请求会在下文介绍），获取一个新的登录态。在服务器返回新的登录态后，客户端需要将其保存在storage中，并且重新调用这个方法，这样做虽然有些繁琐，但是可以保证回调函数中的登录态只会从storage中获取，并且该方法一定会返回一个登录态给调用函数。

向服务器请求新的登录态之前，需要客户端调用小程序的登录接口wx.login（返回值见图8-3）和获取用户信息的接口wx.getUserInfo（返回值见图8-4）。调用登录接口后会得到一个code值，这个值是用用户进行微信登录的凭证，只有通过这个凭证才能拿到真正的微信登录

态。调用获取用户信息的接口会得到用户在微信注册的基本信息。调用这个接口的目的是客户端不知道当前用户是否存在于自己的用户系统中，所以客户端需要向服务器传入微信的用户信息，这样，服务器如果判断出用户不在自己的用户系统时，可以将该用户注册到用户系统。

代码清单8-4 getSessionId

```
getSessionId:function(callback){
    var sessionId = wxService.getStorage('sessionId');
    if(sessionId&&sessionId.length>0){
        if(typeof callback === 'function'){
            callback(sessionId);
        }
    }else{
        weixin.doWxLogin(function(wxLoginRes){//微信登录
            weixin.getWxUserInfo(function(wxUserInfo){//获取微信用户的信息
                var param = {
                    code:wxLoginRes.code,
                    iv:wxUserInfo.iv,
                    signature:wxUserInfo.signature,
                    rawData:wxUserInfo.rawData,
                    encryptedData:wxUserInfo.encryptedData,
                };
                param = us.extend(param,wxUserInfo.userInfo);
                console.log("param",param);
                var url = constant.host + constant.path.getSessionId;
                wx.request({
                    url:url,
                    data:param,
                    complete:function(data){
                        console.log(data);
                        wxService.setStorage('sessionId',data.data.sessionId);
                        weixin.getSessionId(callback);
                    }
                });//end wx.request
            });//end weixin.getWxUserInfo
        });//end weixin.doWxLogin
    }
}
```

```
dowxLogin ▼ Object {errMsg: "Login:ok", code: "011m8m2d2yqUVA010m2d2Bwk2d2m8m2k"} ⓘ
  code: "011m8m2d2yqUVA010m2d2Bwk2d2m8m2k"
  errMsg: "login:ok"
  ▶ __proto__: Object
```

图8-3 wx.login返回的数据

```

encryptedData: "0X0cB9E7CKFeZ4atWd7UnH1XjNDCbkwm1VH04+HA1PldTXfdIKfuhJf0DX2se8Aetj2wCSFhEoiJ19qRAC1+GF9k+YfUF5VgaD0011iRw87C51/j4/8
errMsg: "getUserInfo:ok"
iv: "6HgcNZbqCZRURuI6GPM3UA=="
rawData: "{ \"nickname\": \"边思\", \"gender\": 1, \"language\": \"zh_CN\", \"city\": \"Chengdu\", \"province\": \"Sichuan\", \"country\": \"CN\", \"avatarUrl\": \"http://
signature: \"fbe1865cbeb54b8e3d3bbfa78c6d2bcb7dba5988\"
▼ userInfo: Object
  avatarUrl: \"http://wx.qlogo.cn/mmopen/vi_32/Ir11b0iciaBct1mia8cyDal7mwxPV1Lw3dgTBjHvMSP4SrcfLs41KIRib8ND0C6EKLmGo7fGU7dJnkjpLKZ
  city: \"Chengdu\"
  country: \"CN\"
  gender: 1
  language: \"zh_CN\"
  nickname: \"边思\"
  province: \"Sichuan\"
  __proto__: Object
  __proto__: Object

```

图8-4 wx.getUserInfo返回的数据

下面将介绍本案例登录的后台代码，代码清单8-5是请求/wxapp/getSessionId的代码。这个请求是客户端执行weixin.getSessionId方法时，在用户本地storage中没有登录态的情况下发起的。这个请求的参数就是上文提到的微信用户的基本信息和登录凭证，将这些数据传入userService.getUserWxSessionId方法（下文介绍），在回调中返回客户端新的登录态。

代码清单8-5 请求/wxapp/getSessionId

```

router.get('/wxapp/getSessionId',function(req, res, next){
  var query = req.query;//获取请求参数
  //根据客户端回传的登录信息和用户信息，获取一个新的登录态
  userService.getUserWxSessionId(query,function(newSessionId){
    res.json({//获取登录态成功后，将其返回客户端
      sessionId:newSessionId
    });
  });
  return;
})

```

代码清单8-6是getUserWxSessionId的代码。这段代码首先使用用户的微信登录凭证，通过handle.getWxSession（下文介绍）方法从微信服务器获取该用户在微信的登录态，该登录态包含三个字段：

`sessionKey`、`openId`、`expireDate`。`sessionKey`是微信分配给用户的一个登录标识，这个标识可以用来破解`wx.getUserInfo`方法返回的`encryptedData`中的加密数据。这个`sessionKey`是非常重要的数据，不能直接发送给客户端，需要我们将其保存在自己的数据库中。`openId`是用户针对当前小程序账号（`appId`）的唯一标识，这个字段可以作为我们用户系统与微信用户系统的关联，我们在设计系统时应该考虑根据这个`openId`得到我们用户系统的用户数据。`expireDate`是`sessionKey`的有效时间，一般是30分钟左右，与客户端的登录态的有效时间是一样的，这也是我们验证客户端登录态是否合法的标准。

在继续介绍该方法之前简单介绍下本案例的用户系统。图8-5与图8-6分别存储了用户的基本信息和登录态的信息。用户的基本信息来自于小程序方法`wx.getUserInfo`返回的数据，需要注意`open_id`也在这个表中。除`handle.getWxSession`（下文介绍）返回的数据外，登录态的信息还保存了一个`client_key`，这个值就是客户端的登录态数据，这个数据需要保证严格的加密程序。最后，这两个表是通过`open_id`关联起来的。

再回到`getUserWxSessionId`方法，获取完用户的微信的登录态之后，系统根据`openId`查询`wxapp_user`表，检查这个微信用户是否存在于本地的用户系统中，如果没有，则自动将用户注册在本系统中，然后将登录态保存在`wxapp_user_session`中。如果用户已经存在，则直接根

据openId更新wxapp_user_session中用户登录态信息。

handle.updateSessionKey方法兼容了保存与更新wxapp_user_session表的功能，之后将新的客户端登录态传递给回调函数。

代码清单8-6 getUserWxSessionId

```
getUserWxSessionId:function(data,callBack){
    var code = data.code;
    //向微信服务器获取用户的sessionkey,openId
    handle.getWxSession({code:code},function(wxSessionInfo){
        var openId = wxSessionInfo.openid;
        var sessionKey = wxSessionInfo.session_key;
        data.openId = openId;
        data.sessionId = sessionKey;
        data.expireDate = new Date().getTime()+wxSessionInfo.expires_in/1;
        //根据微信的openId获取本地用户系统中的用户
        handle.getUserByOpenId({openId:openId},function(user){
            if(!user||user.length<=0){//如果用户不存在，则保存
                handle.saveUser(data,function(){
                    //更新sessionKey
                    handle.updateSessionKey(data,function(newSessionId){
                        if(typeof callBack === 'function'){
                            callBack(newSessionId);
                        }
                    });
                });
            }else{//如果用户存在，直接跟新sessionKey
                //更新sessionKey
                handle.updateSessionKey(data,function(newSessionId){
                    if(typeof callBack === 'function'){
                        callBack(newSessionId);
                    }
                });
            }
        });
    });
}
```

名	类型	长度	小数点	允许空值 (
id	bigint	10	0	<input type="checkbox"/>	1
name	varchar	1000	0	<input checked="" type="checkbox"/>	
province	varchar	1000	0	<input checked="" type="checkbox"/>	
city	varchar	1000	0	<input checked="" type="checkbox"/>	
open_id	varchar	1000	0	<input checked="" type="checkbox"/>	
avatar_url	varchar	1000	0	<input checked="" type="checkbox"/>	

图8-5 wxapp_user表

名	类型	长度	小数点	允许空值 (
id	bigint	10	0	<input type="checkbox"/>	1
sessino_key	varchar	1000	0	<input type="checkbox"/>	
open_id	varchar	1000	0	<input type="checkbox"/>	
expire_date	datetime	0	0	<input type="checkbox"/>	
client_key	varchar	1000	0	<input type="checkbox"/>	

图8-6 wxapp_user_session表

代码清单8-7是hande.getWxSession方法，是获取微信登录态的代码。获取微信登录态需要向微信服务器发送一个https请求，地址是<https://api.weixin.qq.com/sns/jscode2session>。这个请求接受四个参数，appid、secret、grant_type、js_code。appid与secret都可以在小程序账号的后台页面中获取（见图8-7），grant_type是固定值authorization_code，js_code是调用wx.login方法返回的code（用户登录凭证）。

代码清单8-7 getWxSession

```
getWxSession:function(data,callBack){
    try{
        var jsCode = data.code;
        if(jsCode){
            var param = {
                appid : constant.appId,//小程序的appId
                secret: constant.secret,//小程序的secretcode
                grant_type:'authorization_code',
                js_code:jsCode
            };
            var content = queryString.stringify(param);
            var option={
                hostname: 'api.weixin.qq.com',
                path: '/sns/jscode2session?' + content,
                method: 'GET'
            };
            var request = https.request(option,function(data){
                data.on('data', function(chunk){
                    var userInfo = JSON.parse(chunk);
                    if(typeof callBack === 'function'){
                        callBack(userInfo);
                    }
                })
            })
        }
    }
}
```

```

});
data.on('error', function(err){
    console.log('RESPONSE ERROR: ' + err);
});
});
request.on('error', function(e) {
    console.log('problem with request: ' + e.message);
});
request.end();
}else{
    console.log("NO code passing");
}
}catch(e){
    console.log(e);
}
}
}

```

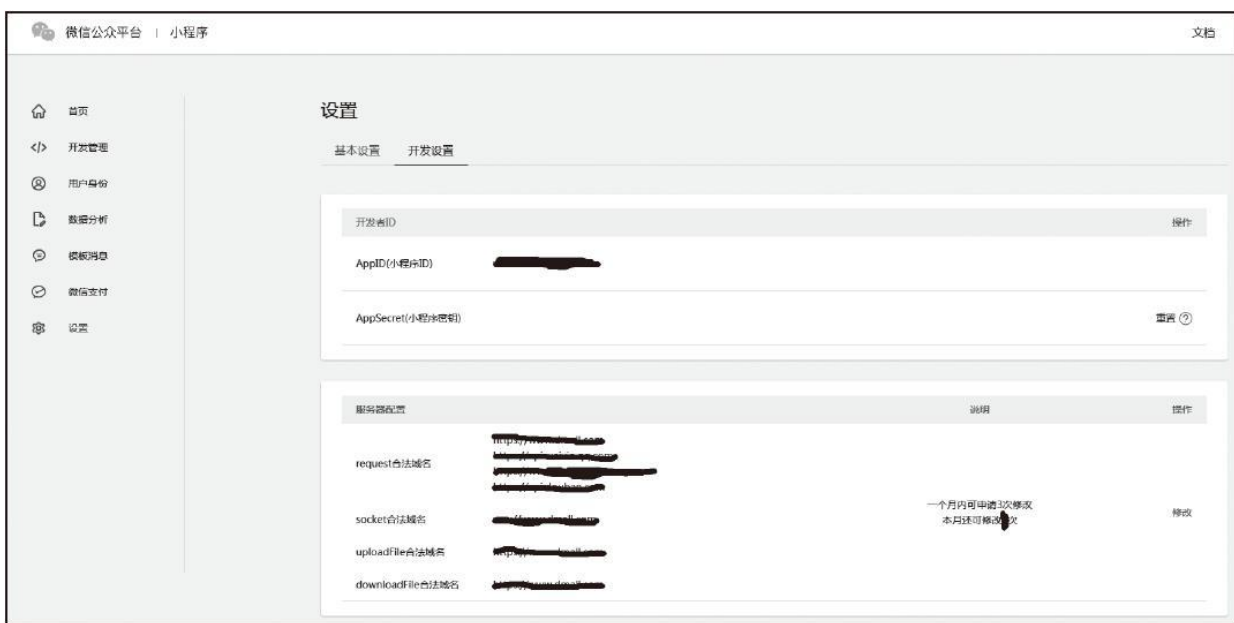


图8-7 获取小程序的appId与secret

8.2 支付

在讲解支付前，先简单介绍下如何开通小程序的微信支付。首先进入小程序账号的后台，来到如图8-8所示的页面。如果未申请过微信支付的话，页面会显示“未开通”。之后开发者就要通过资料审核、账户验证、协议签署三个步骤后才能开通。需要注意的是第一步中需要填写的与企业相关的信息必须与注册小程序账号时填写的信息吻合，否则会导致审核不通过。在审核通过以后用户会得到一个微信商户平台的登录id与密码，使用这个ID与密码进入微信商户平台后，会显示如图8-9所示的页面。安装证书后，根据页面的提示，创建或修改API key。请注意，这个key非常重要，请勿随便告知他人。



图8-8 微信支付申请



图8-9 修改支付API key

回到案例讲解，用户点击打赏按钮后，会进入支付流程，系统会提示用户消费1分钱。这个过程大致可以分为四个阶段。第一个阶段是用户点击打赏按钮，客户端向服务器获取`prepay_id`。用户要通过`wx.requestPayment`调起支付界面，需要传入`prepay_id`。这个参数需要一个相对复杂的流程从微信服务器获取，所以第二个阶段是后台向微信服务器获取`prepay_id`，并将客户端执行`wx.requestPayment`所需的全部参数返回客户端。第三个阶段是前台调起支付界面，供用户发起支付。第四个阶段是在用户支付完成后，等待微信服务器的完成支付回调请求。下边将详细介绍第二阶段的开发。

代码清单8-8是获取prepay_id的主要逻辑，它首先检查用户是否登录，如果没有登录，则返回0001错误码，提示客户端重新登录，获取微信登录态和客户端登录态。如果用户已经登录，则执行getPaymentInfo方法获取prepay_id。

代码清单8-8 请求prepay_id的主要逻辑

```
try{
  //检查用户是否登录
  userService.checkUserSessionId({sessionId:query.sessionId},
  function(result){
    if(!result.isValid){//未登录则要求用户登录
      res.json({
        code:"0001"
      });
    }
    return;
  }
  //获取用户信息
  userService.getUserByCode({code:query.sessionId},
  function(result){
    getPaymentInfo(result);//请求prepay_id
  });
}
}catch(e){
  console.log(e);
}
```

获取微信的prepay_id的方法是向微信服务器发送一个HTTPS请求，请求地址是<https://api.mch.weixin.qq.com/pay/unifiedorder>。这个请求需要传递的参数有许多，具体可以参考官网https://pay.weixin.qq.com/wiki/doc/api/jsapi.php?chapter=9_1。开发者在开发过程中要注意：官网标注的必填字段是必不可少的，但是有些非必填的字段，在一些特殊的支付方式中也是必填的，如本例使用的JSAPI支付方式就要求传递用户的openid，但是这个字段在官网文档上并没有标记为必填。

代码清单8-9是实现获取prepay_id的代码。这段代码首先创建了一个wxpay对象，这个对象是对微信支付抽象出来的一个对象，它的创建需要三个参数，分别为appid、mch_id（商户Id）、partner_key（商户的apikey），在前文中有介绍。这么抽象的原因有两个，第一个原因是在各种微信支付类型中，只有这三个参数是常量；第二个原因是，在正式的开发中，这三个参数应该是保密的，对业务开发者应该是个黑盒，业务方只需要传递其他参数即可。创建完wxpay对象后，调用这个对象的createUnifiedOrder方法，获取prepay_id的请求就是在这个方法中发送的。请求完成后（返回数据见图8-10），如果有数据返回，并且返回的return_code=success以及return_msg=ok时就证明获取prepay_id成功。之后将客户端调起支付页面的参数回传客户端。

代码清单8-9 getPaymentInfo代码

```
var getPaymentInfo = function(userInfo){
    try{
        var wxpay = wxPayService({
            appid: constant.appId,//appid
            mch_id: constant.mchId,//商户Id
            partner_key:constant.apiKey//商户Id密码
        });
        console.log("clientId ",utils.getClientIP(req));
        wxpay.createUnifiedOrder({
            body:constant.paybody,//商品描述
            out_trade_no:wxpay.md5(new Date().getTime()),//内部商品订单号
            total_fee:1,//价格
            spbill_create_ip:utils.getClientIP(req),//终端IP
            notify_url:constant.notifyUrl,//通知地址
            trade_type:'JSAPI',//支付类型
            openid:userInfo.open_id,//用户openId
        },function(prepayInfo){
            var result;
            if(!prepayInfo){//请求失败
                result = {
                    code:'1000',
                    errMsg:'出错啦！'
                };
            }else if(prepayInfo.return_code!='success'){//接口报错
                result = {
```

```

        code: '1001',
        errMsg: prepayInfo.return_msg
    };
} else { // 成功获取prepayId
    result = {
        code: "0000",
        data: { // 前端调起支付框需要的参数
            timeStamp: new Date().getTime(),
            nonceStr: payUtil.generateNonceString(),
            package: prepayInfo.prepay_id,
            signType: "MD5",
        }
    };
    paySign = wxpay.sign(result); // 生成签名
    result.paySign = paySign;
}
res.json(result);
});
} catch (e) {
    console.log(e);
}
};

```

```

<xml><return_code><![CDATA[SUCCESS]]></return_code>
<return_msg><![CDATA[OK]]></return_msg>
<appid><![CDATA[ ]]></appid>
<mch_id><![CDATA[ ]]></mch_id>
<nonce_str><![CDATA[d17RLkJTSsCK8SSQ]]></nonce_str>
<sign><![CDATA[AA8577B5E5347706CDF2DDE36605A92B1]]></sign>
<result_code><![CDATA[SUCCESS]]></result_code>
<prepay_id><![CDATA[wxB1611252050469a8dde33b00013455577]]></prepay_id>
<trade_type><![CDATA[JSAPI]]></trade_type>
</xml>

```

图8-10 成功获取prepay_id

代码清单8-10是生成签名的代码。在获取prepay_id的开发中，有一个重要的环节就是生成签名，笔者也是在生成签名的地方纠结了很久。生成签名的具体步骤请参考官网的介绍

https://pay.weixin.qq.com/wiki/doc/api/jsapi.php?chapter=4_3，在此只通过一个简单的例子说明这个步骤，以及开发者在开发过程中遇到微信服务器返回“签名错误”时应该如何排查问题。

代码清单8-10 生成签名的代码

```
WXPay.mix('sign', function(param){
    var querystring = Object.keys(param).filter(function(key){
        return param[key] !== undefined && //排除值为空的参数
        param[key] !== '' &&
        ['pfx', 'partner_key', 'sign', 'key'].indexOf(key)<0; //排除不参与排序的参数
    }).sort().map(function(key){ //排序并且生成url参数格式
        return key + '=' + param[key];
    }).join("&") + "&key=" + this.options.partner_key; //最后将apikey放在后边
    return md5(querystring).toUpperCase(); //MD5加密并转大写
});
```

代码清单8-11到8-15以一个模拟的实例说明了在支付方式为JSAPI的情况下生成签名的过程。遇到微信服务器返回的错误信息为签名错误时，可以通过微信官网提供的签名检测工具进行校验（如图8-11）。如果发现工具返回的签名与自己代码生成的签名不一致，则有可能是自己的签名算法出了问题，如果工具返回的签名与自己代码生成的一致，则开发者应该考虑参数是否有不合法的情况，如appid或者apikey不正确等。

代码清单8-11 模拟请求数据

```
{
  body: '多点生鲜-网上好超市',
  out_trade_no: 'ec15ba59d1a43b4d9fbe0d238cc96fe2',
  total_fee: 1,
  spbill_create_ip: '127.0.0.1',
  notify_url: 'wx.dmall.com',
  trade_type: 'JSAPI',
  openid: 'openId',
  nonce_str: '1bq207YZ8uasVr6IbXd1v5knLq9XwRXq',
  appid: 'appid',
  mch_id: 'mch_id',
}
```

代码清单8-12 字典序排列后的数据

```
{
  appid: 'appid',
  body: '多点生鲜-网上好超市',
}
```

```
mch_id: 'mch_id',
nonce_str: '1bq207YZ8uasVr6IbXd1v5knLq9XwRXq',
notify_url: 'wx.dmall.com',
openid: 'openId',
out_trade_no: 'ec15ba59d1a43b4d9fbe0d238cc96fe2',
spbill_create_ip: '127.0.0.1',
total_fee: 1,
trade_type: 'JSAPI',
}
```

代码清单8-13 URL参数化的数据

```
appid=appid&body=多点生鲜-网上好超市
&mch_id=mch_id&nonce_str=1bq207YZ8uasVr6IbXd1v5knLq9XwRXq&notify_url=wx.dmall.com&
openid=openId&out_trade_no=ec15ba59d1a43b4d9fbe0d238cc96fe2&spbill_create_ip=127.0
.0.1&total_fee=1&trade_type=JSAPI
```

代码清单8-14 添加APIKEY的数据

```
appid=appid&body=多点生鲜-网上好超市
&mch_id=mch_id&nonce_str=1bq207YZ8uasVr6IbXd1v5knLq9XwRXq&notify_url=wx.dmall.com&
openid=openId&out_trade_no=ec15ba59d1a43b4d9fbe0d238cc96fe2&spbill_create_ip=127.0
.0.1&total_fee=1&trade_type=JSAPI&key=key
```

代码清单8-15 通过MD5算法得到的数据

```
5F62CF23F03920456FF32862CCD7E801
```

8.3 小结

本章实例主要讲解了两个功能点：

1) 小程序的登录API以及后台用户系统的开发，读者可以了解到如何将自己的用户系统与微信的用户系统结合，实现使用微信账号登录本系统的功能。

2) 小程序的支付API以及微信的支付API的使用，读者可以了解使用微信支付的基本流程。

增加参数

生成签名

参数名称：	<input type="text" value="appid"/>	参数值：	<input type="text" value="appid"/>	删除
参数名称：	<input type="text" value="mch_id"/>	参数值：	<input type="text" value="mch_id"/>	删除
参数名称：	<input type="text" value="body"/>	参数值：	<input type="text" value="多点生鲜-网上好超市"/>	删除
参数名称：	<input type="text" value="nonce_str"/>	参数值：	<input type="text" value="1bq207YZ8uasVr6IbXd1v5knLq9XwRXq"/>	删除
参数名称：	<input type="text" value="notify_url"/>	参数值：	<input type="text" value="wx.dmall.com"/>	删除
参数名称：	<input type="text" value="openid"/>	参数值：	<input type="text" value="openid"/>	删除
参数名称：	<input type="text" value="out_trade_no"/>	参数值：	<input type="text" value="ec15ba59d1a43b4d9fbe0d238cc96fe2"/>	删除
参数名称：	<input type="text" value="spbill_create_ip"/>	参数值：	<input type="text" value="127.0.0.1"/>	删除
参数名称：	<input type="text" value="total_fee"/>	参数值：	<input type="text" value="1"/>	删除
参数名称：	<input type="text" value="trade_type"/>	参数值：	<input type="text" value="JSAPI"/>	删除
商户Key:	<input type="text" value="key"/>			

增加参数

生成签名

#1.对参数按照key=value的格式，并按照参数名ASCII字典序排序生成字符串：

appid=appid&body=多点生鲜-网上好超市&mch_id=mch_id&nonce_str=1bq207YZ8uasVr6IbXd1v5knLq9XwRXq¬ify_url=wx.dmall.com&openid=openid&out_trade_no=ec15ba59d1a43b4d9fbe0d238cc96fe2&spbill_create_ip=127.0.0.1&total_fee=1&trade_type=JSAPI

#2.连接商户key：

appid=appid&body=多点生鲜-网上好超市&mch_id=mch_id&nonce_str=1bq207YZ8uasVr6IbXd1v5knLq9XwRXq¬ify_url=wx.dmall.com&openid=openid&out_trade_no=ec15ba59d1a43b4d9fbe0d238cc96fe2&spbill_create_ip=127.0.0.1&total_fee=1&trade_type=JSAPI&key=key

#3.md5编码并转成大写：

sign=5F62CF23F03920456FF32862CCD7E801

图8-11 微信签名校验工具

第9章 案例分析——日程表

随着社会节奏的变快，人们的生活内容变得更加充实，我们在享受丰富多彩生活的同时，也饱受手忙脚乱带来的烦恼，所以一款基于日程管理功能的App便是解救“大忙人”的日常必备。本章的练习项目是日程表，该项目的主要功能是计划和管理自己的日程。通过本章的学习，读者可以进一步了解如何开发和设计复杂的基于微信小程序的App。本项目的源码地址为<https://github.com/wxapp-book/calendar.git>。

9.1 业务流程

日程表App包括3个页面：首页、日程详情页、管理页。下面将分别介绍各个页面的主要功能。

首页（见图9-1）：系统的入口页面，分为三个区域：日期区域，日历区域和日程区域。日期区域默认显示当天的日期，点击后可以进入“日程管理”页面。日历区域显示当月的日历，用户可以点选日历上的日期刷新日期区域的显示信息。日程区域提供操作当天日程的功能。进入页面后，系统会自动从缓存中读取当天的日程安排，读完后会根据这些日程的开始时间、结束时间，以及用户是否标记该日程已经结束，将其分为三个列表：进行中日程列表、未开始日程列表、已结束日程列表。下面是列表说明：

2016年

+

16 星期五
11月

SCHEDULE

日	一	二	三	四	五	六
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

早报 结束10:23	一般
---------------	----

早会 结束10:23	重要
---------------	----

品牌设计方案讨论 结束10:23	一般
---------------------	----

微信商城2.1需求整理 结束10:23	一般
------------------------	----

图9-1 首页

·“进行中日程”列表包含了当前时间晚于日程开始时间的日程，但是并不限制当前时间早于日程结束时间。这个列表会把日程的结束时间显示在界面上。用户点击这些日程后，页面底部会弹出浮层，用户可以选择更新日程，跳转的日程详情页修改日程信息。也可以选择完成日程，将该日程从进行中日程列表中移除。

·“未开始日程”列表包含了当前时间早于日程开始时间的日程。这个列表会把日程的开始时间显示在界面上。用户点击这些日程后，页面底部会弹出浮层，用户只有一个选项就是“修改日程”，点击后跳转日程详情页修改日程信息。

·“已结束日程”列表包含了用户在“进行中日程列表”中通过“完成日程功能”从该列表中移除的日程。该列表不会展示日程的开始时间或者结束时间，用户点击后也不会有浮层弹出。用户创建的日程在这个列表中完成生命周期的闭环。

日程详情页（见图9-2）：查看、修改、创建日程。日程详情页有三个入口，首页有两个，分别为创建当天的日程和修改当天的日程。日程管理页有一个入口，作用是创建不同日期的日程。日程详情页在创建或修改日程后会自动返回调用页面。

日程管理页面（见图9-3）：页面由两个区域组成，日历区域和日程区域。日程管理页只有一个入口，首页的日期区域。用户点击该区域后，会将当时显示的日期传入日程管理页面作为日程管理页面日历区域的首选日期，并且以该日期为准，向后展示七天作为可选日期，用户点击日期后，会刷新下方的日程管理区域，并且用户点击页头的添加日程按钮，会进入日程详情页为该日期添加日程的。日程管理区域是一个表格，一共有24行，代表了一天的24个小时，表格的列数是根据该日期日程的多少及日程的时间分布动态生成的，具体的逻辑会在下面功能点分析小节详细介绍。

取消

编辑事件

完成

标题

优先级

一般 >

全天

☐

开始

2016年12月16日 周日 14:00

结束

2016年12月16日 周日 15:00

删除事件

图9-2 日程详情页



图9-3 日程管理页

9.2 项目架构

9.2.1 功能点分析

本项目的开发难点主要有两个，第一是如何存储日程数据，第二是日程管理页面的日程区域。

存储日程数据：本章的练习项目是一个纯本地项目，没有与服务器的交互，所以，日程数据都需要存储在小程序的**storage**中。由于系统需要经常添加，修改和查询日程，所以一套高效和稳定的日程存储方案就是保证系统稳定的前提。

存储一共涉及了两个对象，日期对象和日程对象。这两个对象的键值分别以**date_**和**task_**开头，日期对象会以被创建日期零点的毫秒值作为结尾，而日程对象则会以当前时间的毫秒值作为结尾，这样，就保证了日期对象和日程对象的唯一性。

在创建日程之前，需要检查**storage**中是否保存了当前的日期对象，因为日程需要与日期关联起来才会在首页和日程管理页被查询出来。而在日期对象中，保存一个关联了当日所有日程的列表，每当创建完日程后，需要将该日程的**id**添加到这个列表中。

日程管理页面的日程区域：这个区域是一个表格，一共有24行，代表了一天的24个小时，表格的列数根据该日期日程的多少及日程的时间分布动态生成。在向表格添加日程的时候，会优先添加到最左边的列，如果该列的相关时间段没有其他日程，则该日程会被添加到该列，如果该列的相关时间段已有其他日程，则需检查右相邻列的相关时间段是否有其他日程，如果所有列相关时间段都不满足添加的条件，则创建一个新列添加这个日程。例如图9-3所示，第一列与第二列，都有一个10：41到11：41的日程，如果这时再添加一个该时间段内的日程，则这两列都没有办法显示这个日程，需要再创建一个列去存放这个日程。

9.2.2 项目结构图

项目结构如图9-4所示。





























- ▼  calendar
 - ▼  common
 - ▶  css
 - ▼  js
 -  constant.js
 -  service.js
 -  wx.js
 - ▼  lib
 -  moment.js
 -  underscore.js
 - ▼  pages
 - ▼  create
 -  create.js
 -  create.wxml
 -  create.wxss
 - ▼  daytask
 -  daytask.js
 -  daytask.wxml
 -  daytask.wxss
 - ▼  index
 -  index.js
 -  index.wxml
 -  index.wxss
 - ▼  utils
 -  util.js
 -  app.js
 -  app.json
 -  app.wxss

图9-4 项目结构图

下面介绍各个部分所承担的功能。

`common`存放一些公共业务相关的代码，其中：

- `wx.js`：封装一些微信小程序的底层API。在开发过程中，有一些底层的小程序API需要我们对自身业务进一步封装的。

- `constant.js`：保存系统的常量，例如一些常用的storage key。

- `service.js`：向外提供了dateService和taskService两个类，这两个类的作用是向业务代码提供了保存、修改、读取日期和日程的功能代码。

`lib`存放第三方的类库，其中：

- `underscore.js`：微信小程序的开发是基于数据模型的，所以，会有大量的数据操作，比如列表查询，数据格式转换等工作。在这里选择underscore.js作为数据处理的工具，他有很多优势如十分小巧，压缩后只有4KB；使用方法简单，它提供了几十种函数式编程的方法，大大方便了JavaScript的编程；覆盖面广，包含了操作集合、数组、函数和对象的方法；社区人气高，不用担心技术支持的问题。

·**moment.js**: 一个功能非常强大的时间操作工具。在项目开发的过程中，会大量涉及日期的操作，如改变日期的输出格式，获取下一天，获取某星期、某月的第一天和最后一天，获取某一天的开始时间的毫秒和结束时间的毫秒等，这些功能如果全部自己开发会非常耗时，而且这些功能不是我们关注的重点，所以选择了**moment.js**作为日期的工具库，以便我们更加专注业务的实现。

pages存放页面代码，其中：

·**create**: 日程详情页

·**daytask**: 日程管理页。

·**index**: 首页。

9.3 代码分析

在日程App里，首页与日程管理页都需要依赖日程数据，而日程详情页不需要依赖任何数据，并且日程详情页可以用来创建日程数据，所以开发时选择该页面作为入手点。日程管理页需要依赖首页为其传入参数，而且日程管理页的开发难度较高，所以日程管理页放在了最后一个去开发。本章节的讲解顺序也会按照这个思路展开。

9.3.1 日程详情页

日程详情页主要用于创建和修改日程，所有与日程有关的数据都会出现在这个页面，图9-5是这个页面的数据模型。下面对数据模型中的重要字段进行分析。

·**curDate**: 当前日程的**moment**对象。在进入日程详情前，用户需要选择日期或者选择一个日程。所以本页面只需使用上级页面传递过来的日期或者当前日期即可。

·**pageType**: 用来标记当前页面是被用作创建日程还是被用作修改日程，该字段有两个可选值：**create**和**update**。

·**task**: 创建或者修改的日程数据，同时也是本页面渲染需要的数据，包括开始时间，结束时间，重要性，日程内容等。这个对象最后会被保存到**storage**中作为日程数据供首页和日程管理页调用。

·**taskImportant**: 日程重要性选项。日程重要性的选择会通过小程序的**picker**组件去实现，这个字段中的数据会作为这个组件的参数被使用。

```

> getCurrentPages()[1].data
< ▼ Object {__webviewId__: 2, curDate: Moment, task: Object, taskTime: Object, taskImportant: Array[2]-} ⓘ
  __webviewId__: 2
  ▶ curDate: Moment
  pageType: "create"
  ▼ task: Object
    date: "2016-12-18"
    endTime: "14:56"
    endTimeMs: 1482044160000
    important: "一般"
    startTime: "13:56"
    startTimeMs: 1482040560000
    title: "新建任务"
    ▶ __proto__: Object
  ▼ taskImportant: Array[2]
    0: "一般"
    1: "重要"
    length: 2
    ▶ __proto__: Array[0]
  ▼ taskTime: Object
    date: "2016-12-18"
    endTime: "14:56"
    endTimeBeginLimit: "13:56"
    startTime: "13:56"
    startTimeBeginLimit: "13:56"
    ▶ __proto__: Object
  ▶ __proto__: Object

```

图9-5 日程详情页-数据模型

·**taskTime**: 为了减少交互的复杂度，用户选择日期的组件同样选择的picker，这个组件在作为时间选择器的时候，可以指定可选时间的范围，通过设置这个范围，就避免了每次提交日程时去验证日期是否合法的麻烦。**taskTime**这个对象就是用来渲染开始时间和结束时间的对象。用户每次进入该页面或者切换开始时间时，都会触发这个对象的更新，然后刷新picker组件的行为。

代码清单9-1是日程详情页加载的代码，这段代码的主要工作就是初始化上文提到的五个页面模型字段。首先，传入该方法的参数可能有两个，**pageType**和**key**。当**pageType**为**create**时不会有**key**传入，会初始化一个默认的**task**对象。而当**pageType**为**update**时，就会有**key**对象传

入，这时就需要根据这个key从storage中获取对应的task对象供页面渲染所用。

代码清单9-1 日程详情页加载

```
onLoad:function(option){
    var pageType = option.pageType||'create';
    var task;
    var curDate;
    if(pageType === 'create'){
        var ms = option.ms || new Date().getTime();
        curDate = moment(ms,"x");
        task = {
            title:"新建日程",
            important:"一般",
            date:moment(ms,'x').format("YYYY-MM-DD")
        };
    }else{
        var key = option.key;
        if(key){
            task = taskService.get({key:key});
            curDate = moment(task.startTimeMs);
        }
    }
    this.setData({curDate:curDate});
    var taskTime = _fn.getTaskTime(task);
    var taskImportant = ['一般','重要'];
    this.setData({
        task:task,
        taskTime:taskTime,
        taskImportant:taskImportant,
        pageType:pageType,
    });
}
```

代码清单9-2是刷新taskTime对象的方法。这个方法会在用户修改开始日期、结束日期或者选择是否全天日程时被调用。这段代码的主要逻辑是，在刚进入页面时，开始时间和结束时间分别为当前时间和当前时间向后推移一个小时，在用户修改开始时间后，如果结束时间早于开始时间，则将结束时间设置为开始时间向后推移一个小时。日程开始时间的限制始终都不能早于当前时间，日程结束时间的限制为始终都不能早于日程开始时间。

代码清单9-2 刷新taskTime对象方法

```
getTaskTime:function(task){
    task = task || {};
    var now = utils.getPageData().data.curDate;
    var dateStr = task.date || now.format('YYYY-MM-DD');
    var curTask = _fn.getCurTask()||task||{};
    var startTime = curTask.startTime?curTask.startTime:now.format("HH:mm");
    var startTimeMoment = moment(dateStr+" "+startTime);
    var endTimeMoment;
    if(curTask.endTime){
        endTimeMoment = moment(dateStr+" "+curTask.endTime);
        if(!endTimeMoment.isAfter(startTimeMoment)){
            endTimeMoment = moment(startTimeMoment);
            endTime = endTimeMoment.add(1, 'h').format("HH:mm");
        }else{
            endTime = curTask.endTime;
        }
    }else{
        endTimeMoment = moment(startTimeMoment);
        endTimeMoment = endTimeMoment.add(1, 'h');
        endTime = endTimeMoment.format("HH:mm");
    }
    curTask.startTimeMs = startTimeMoment.valueOf();
    curTask.endTimeMs = endTimeMoment.valueOf();
    curTask.startTime = startTime;
    curTask.endTime=endTime;

    var startTimeBeginLimit = now.format("HH:mm");
    var endTimeBeginLimit = startTime;

    var taskTime = {
        startTime:startTime,
        endTime:endTime,
        startTimeBeginLimit:startTimeBeginLimit,
        endTimeBeginLimit:endTimeBeginLimit,
        date:dateStr
    };
    return taskTime;
}
```

代码清单9-3是用户点击添加或者修改后执行的代码，这里调用taskService的create和update方法去保存日程数据，保存成功后，返回上级页面。这里使用了同步的方法去保存日程数据，是因为保存完成后会马上回到上级页面，上级页面会立即读取storage中的日程数据。这样就需要保证缓存中的数据是包含新添加的日程数据的。如果采用异

步的方式，是无法保证这点的，导致首页或者日程管理页上将看不到这条新添加的日程数据。

代码清单9-3 保存日程事件/修改日程事件

```
saveTask:function(e){
    var task = _fn.getCurTask();
    taskService.creat(task);
    wx.navigateBack({
        delta:1
    });
}
updateTask:function(e){
    var taskKey = e.target.dataset.taskkey;
    var task = _fn.getCurTask();
    taskService.update({
        key:taskKey,
        val:task
    });
    wx.navigateBack({
        delta:1
    });
}
```

代码清单9-4是在service.js中的保存日程代码。这段代码可以分成三个部分，第一个部分是获取date对象，第二个部分是保存日程对象，第三个部分是向日期对象添加日程对象。下边介绍这三个部分中涉及的具体代码逻辑。

代码清单9-4 保存日程

```
creat:function(task){
    //第一部分：获取时间对象
    var ms = task.startTimeMs;//日程开始时间的毫秒值
    //读取该毫秒对应缓存的日期对象
    var date = dateService.get({ms:ms});
    var dateKey;
    if(!date){//缓存无日期对象则创建
        dateKey = dateService.create({ms:ms});
        date = dateService.get({key:dateKey});
    }else{
        dateKey = date.key;
    }
    //第二部分：保存日程对象
```

```
var taskKey = taskService.getTaskKey(moment().valueOf());
task.key = taskKey;
wxService.setStorage({
  key:taskKey,
  val:task
});
//第三个部分：向日期对象添加日程对象
date.taskKeys = date.taskKeys || [];
date.taskKeys.push(taskKey);
dateService.update({
  key:dateKey,
  val:date
});
}
```

在第一部分中，需要获取storage中日期对象的key值，日期对象的key值规定：以date_开头，以某日期零时的毫秒值作为结尾。在代码清单9-4中，通过参数中的毫秒值，得到对应的moment对象，又通过moment对象的startOf方法，得到对应日期零时的毫秒值。保存日期时获取key值，也是通过这个方法得到一个key之后使用这个key保存日期对象的。只要保证保存时和读取时，传入该方法的毫秒值在同一天，就可以得到相同的key值，如下代码所示。

```
getDateKey:function(ms){
  var dateKey = 'date_'+moment(ms).startOf('day').valueOf();
  return dateKey;
}
```

在第二部分中，需要生成日程的key，日程对象的key值规定，以task_开头，以创建时间的毫秒值作为结尾。通过这个方法，就可以保证日程对象在storage中的唯一性，如下所示：

```
getTaskKey:function(ms){
  var taskKey = 'task_'+ms;
  return taskKey;
}
```

图9-6是date对象和task对象在storage中存储的截图。可以发现，日期对象中有一个列表对象taskKeys。这个对象就是映射该日期下关联的日程对象的。保存日程的第三部分就是将新生成的日程对象的key值添加到该对象中。

logs	Array [1482047145071, 1482040555086, 1481955425927, 1481945383546, 1481945360555, 1481943570861, 14819424247110]
date_1481904000000	{ "key": "date_1481904000000", "taskKeys": ["task_1481942451720", "task_1481942464060", "task_1481942498839", "task_1481942503220", "task_1481942512584", "task_1481942527187"], "startTime": 1481904000000, "endTime": 1481900000000 }
task_1481942451729	{ "title": "测试任务1", "important": "一般", "date": "2016-12-17", "startTime": 1481942400000, "endTime": 1481943000000, "startTime": "10:40", "endTime": "11:40", "key": "task_1481942451729", "status": "finish" }
task_1481942464060	{ "title": "测试任务2", "important": "一般", "date": "2016-12-17", "startTime": 1481942400000, "endTime": 1481943000000, "startTime": "10:40", "endTime": "11:40", "key": "task_1481942464060", "status": "finish" }
task_1481942498839	{ "title": "测试任务6", "important": "一般", "date": "2016-12-17", "startTime": 1481939900000, "endTime": 1481940450000, "startTime": "14:41", "endTime": "15:41", "key": "task_1481942498839" }
task_1481942503220	{ "title": "测试任务5", "important": "一般", "date": "2016-12-17", "startTime": 1481964000000, "endTime": 1481967580000, "startTime": "15:41", "endTime": "17:41", "key": "task_1481942503220" }
task_1481942512584	{ "title": "测试任务4", "important": "一般", "date": "2016-12-17", "startTime": 1481942400000, "endTime": 1481943000000, "startTime": "10:41", "endTime": "11:41", "key": "task_1481942512584" }
task_1481942527187	{ "title": "测试任务3", "important": "一般", "date": "2016-12-17", "startTime": 1481942400000, "endTime": 1481943000000, "startTime": "10:41", "endTime": "11:41", "key": "task_1481942527187" }

图9-6 storage中的date对象和task对象

9.3.2 首页

首页的内容可以分为三个区域，日期区域、月历区域和日程区域。图9-7是首页的数据模型，下面对其中的重要字段作解释：

- calendar**: 渲染当月日历的字段。这个字段保存了一个星期维度和日期维度的二维数组。

- days**: 显示星期几的映射字段。

- groupTask**: 日程区域的渲染字段。日程区域会将当日的日程分成三个部分，未完成的日程，完成的日程和未开始的日程。

- selectDate**: 日期区域的渲染字段。每次用户选择月历区域的日期时，都会刷新这个字段的数据。

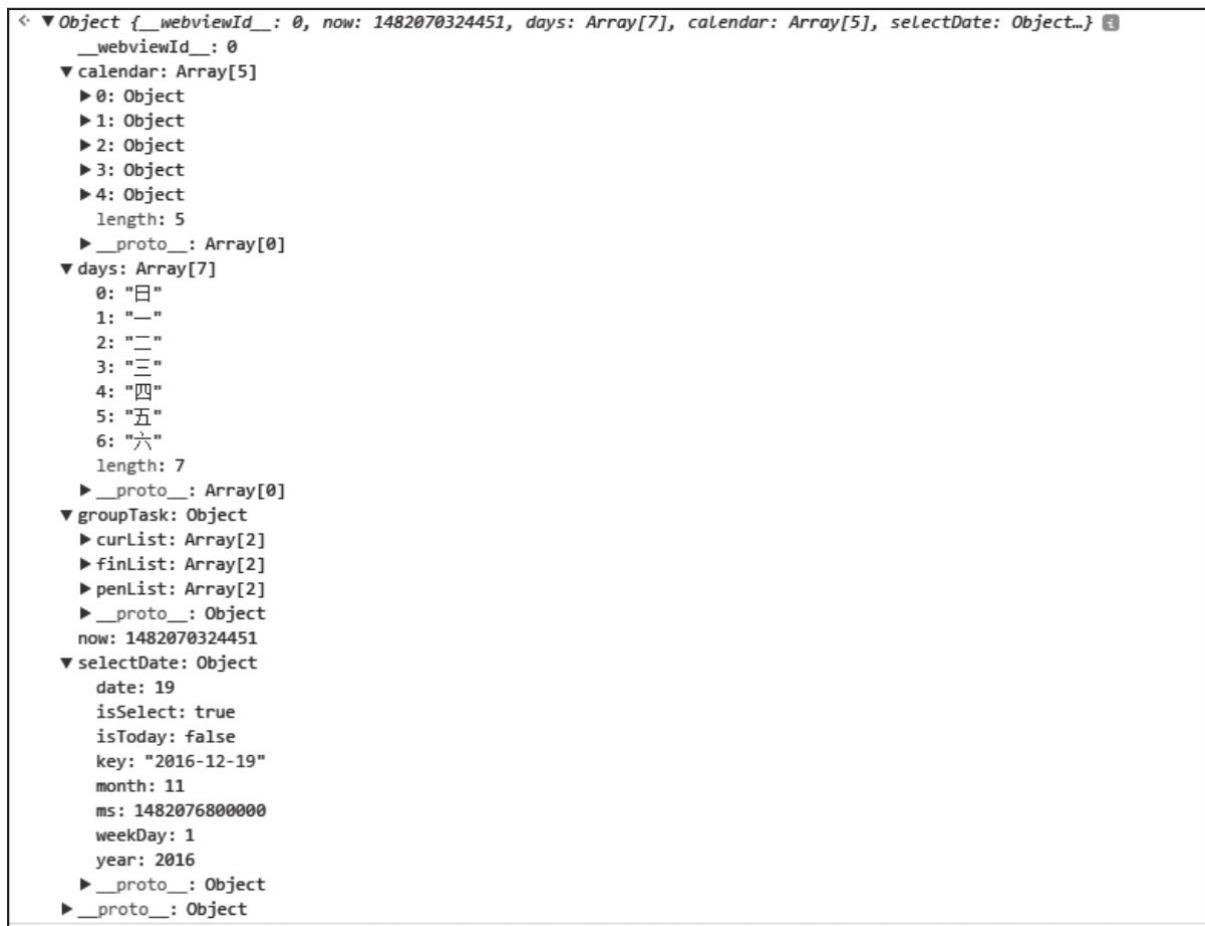


图9-7 页面模型数据

代码清单9-5是首页加载的代码，这段代码的主要逻辑是初始化页面数据模型中的字段。这里调用了小程序的两个生命周期函数：**onLoad**和**onShow**方法。**onLoad**方法只会被调用一次，所以这段代码只加载了日历和当天的日期数据。**onShow**方法会在每次进入这个页面时被调用。当该页面是从日程管理页或者日程详情页回来时，当天的日程数据有可能发生变化，为了与**storage**中的日程数据同步，需要每次

进入页重新读取并生成groupTask字段，onShow中的_fn.init方法就是执行这个过程的。

代码清单9-5 页面加载

```
onLoad: function () {  
    // wxService.clearStorage();  
    _fn.getCurPage().setData({  
        now:new Date().getTime(),  
        days:constant.calendar.dayShort,  
        calendar:calendar.getCalendarData('m'),  
        selectDate:calendar.getToday()  
    });  
},  
onShow:function(){  
    _fn.init();  
}
```

代码清单9-6是获取页面模型groupTask的字段。由于首页的日程区域只会显示当天的日程，所以只需传递当天一个毫秒值。在获取到当天的日程列表后，将这个日程列表按照其状态分成三个列表，分别为未开始、未完成和已完成。然后将未开始的日程按照开始时间升序排列，未完成的日程按照结束时间的升序排列，已结束的日程按照结束时间的降序排列。

代码清单9-6 获取渲染日程区数据

```
groupTask:function(){  
    var ms = new Date().getTime();  
    taskService.getDayTasks({ms:ms}, function(taskList){  
        var penList =  
taskService.filterTaskByStatus(taskList,constant.taskStatus.pending);  
        var curList =  
taskService.filterTaskByStatus(taskList,constant.taskStatus.current);  
        var finList =  
taskService.filterTaskByStatus(taskList,constant.taskStatus.finish);  
  
        penList = taskService.orderTaskByStartTime(penList,constant.orderType.asc);  
        curList = taskService.orderTaskByEndTime(curList,constant.orderType.asc);  
        finList = taskService.orderTaskByEndTime(finList,constant.orderType.desc);  
    });  
}
```

```

var groupTask = {
    penList:penList,
    curList:curList,
    finList:finList
};
_fn.getCurPage().setData({
    groupTask:groupTask
});
});
},

```

代码清单9-7是获取某日期日程数据的方法，这个方法位于service.js的taskService对象内部。这个方法是一个公共服务，主要实现了通过一个毫秒值获取该毫秒对应日期内用户创建的日程，同时该方法提供了同步方式调用和异步方式调用，区别在于是否传递了callBack参数。方法首先根据毫秒读取storage中的日期对象，接着遍历该对象中的taskKeys数组从而得到该日期下的所有日程key，然后通过同步的方式得到日程对象并将其保存在一个列表中。

代码清单9-7 获取某日期的日程数据

```

getDayTasks:function(option,callBack){
    var ms = option.ms;
    var getTasks = function(date){//根据日期对象中的taskKeys对象获取日程对象
        var taskKeyList = date.taskKeys||[];
        var taskList = [];
        for(var i = 0 ; i < taskKeyList.length ; i++){
            taskList.push(taskService.get({key:taskKeyList[i]}));
        }
        if(callBack && typeof callBack==='function'){
            callBack(taskList);
        }else{
            return taskList
        }
    };
    if(callBack && typeof callBack==='function'){//异步
        dateService.get({ms:ms},function(result){//从storage中得到ms对应日期对象
            getTasks(result);
        });
    }else{
        var taskList = dateService.get();//同步
        return getTasks(taskList);
    }
}

```

代码清单9-8是根据日程状态过滤日程列表的代码，这个方法位于service.js的taskService对象内部。为了方便用户了解当日日程的进展程度，在系统内部为日程定义了三个状态，分别为未开始、未结束和已结束，定义位于constant.js内部。该方法通过调用js的数组原生方法Array.filter去过滤参数taskList。已结束的日程是在日程对象上添加status=finish来实现的。未开始和未结束的日程则是根据当前时间与日程开始时间的前后关系来决定，在代码中这个判断是借助momentjs的isBefore方法和isAfter方法来实现的。

代码清单9-8 根据日程状态过滤日程列表

```
filterTaskByStatus:function(taskList,status){
    var returnList = taskList;
    returnList = taskList.filter(function(a){
        var momentStart = moment(a.startTimeMs);//日程开始时间的moment对象
        var momentEnd = moment(a.endTimeMs);//日程结束时间的moment对象
        var now = moment();
        var taskStatus = a.status;
        if(constant.taskStatus.pending===status){
            if(now.isBefore(momentStart)){//当前时间早于开始时间
                return true;
            }
        }else if(constant.taskStatus.current===status &&taskStatus !==
constant.taskStatus.finish){
            if(now.isAfter(momentStart)){//当前时间晚于开始时间，并且用户未标记日程结束
                return true;
            }
        }else if(constant.taskStatus.finish===status){
            if(taskStatus === constant.taskStatus.finish){//用户标记日程结束
                return true;
            }
        }
    });
    return returnList;
}
```

代码清单9-9是计算日历数据的代码。这段代码位于utils.js的calendar对象中。方法的入口为getCalendarData，获取日历的类型分为

按周获取和按月获取。按周获取月历的方法是`getWeekData`，该方法首先获取参数毫秒值对应周的开始日期和结束日期，使用的方法是`momentjs`的`startOf`和`endOf`方法，同时声明一个基准日期为周开始时间，接下来便是将这个基准日期按天相加直到周结束日期，每次相加后会判断相加后的日期是否为今天并且将其标记，返回的结果是一个长度为7的一维数组。按月获取月历的方法是`getMonthData`，这个方法与`getWeekData`方法的思路类似，不同的是首先获取参数毫秒值对应月的开始日期和结束日期，日期递增的方式不是一天而是七天，最后递增到大于或者等于月结束日期，最后返回的数据是当月的所有周的数据。由于月历数据是按照周去计算日期，所以难免会包含不属于当月的日期，所以在使用月历数据渲染页面的时候需要隐藏这些不正确的日期，这里使用了`font-size=0`的方式去处理，从而减少了数据处理和页面渲染的难度。

代码清单9-9 计算日历数据

```
//获取月历对象的入口函数
getCalendarData:function (calendarType,data){
    var now = data || moment();
    var calendarData;
    if(calendarType === 'm'){//获取一个月的月历
        calendarData = calendar.getMonthData(now);
    }else if(calendarType === 'w'){//获取一个星期的月历
        calendarData = calendar.getWeekData(now);
    }
    return calendarData;
},
//获取一周的月历
getWeekData:function(time){
    //周开始日期
    var weekStart = moment(time).startOf('week');
    //周结束日期
    var weekEnd = moment(time).endOf('week');
    var loopMoment = weekStart;//基准日期
    var process = true;
```

```

var weekDays = [];
while(process){
  weekDays.push({
    year:loopMoment.year(),
    month:loopMoment.month(),
    date:loopMoment.date(),
    weekDay:loopMoment.weekday(),
    key:loopMoment.format('YYYY-MM-DD'),
    ms:loopMoment.valueOf(),
    isToday:loopMoment.format('YYYY-MM-DD')===moment().format('YYYY-MM-DD'),
    isSelected:loopMoment.format('YYYY-MM-DD')===moment().format('YYYY-MM-DD')
  });
  //如果基准日期等于或者超过周结束日期，则停止循环，否则加一天
  if(loopMoment.isSame(weekEnd, 'day')||loopMoment.isAfter(weekEnd)){
    process=false;
  }else{
    loopMoment.add(1, 'day');
  }
}
return weekDays;
},
//获取一个月的月历
getMonthData:function(time){
  //月开始日期
  var monthStart = moment(time).startOf('month');
  //月结束日期
  var monthEnd = moment(time).endOf('month');
  var loopMoment = monthStart;//基准日期
  var process = true;
  var monthWeeks = [];
  while(process){
    //获取一个星期的数据
    var weeks = calendar.getWeekData(loopMoment);
    monthWeeks.push({
      key:loopMoment.format('YYYY-W'),
      month:loopMoment.month(),
      weeks:weeks
    });
    //将基准数据加七天，如果等于或者超过月结束日期，则停止循环
    loopMoment.add(7, "day");
    if(loopMoment.isSame(monthEnd, 'day')||loopMoment.isAfter(monthEnd)){
      process = false;
    }
  }
  return monthWeeks;
}
}

```

代码清单9-10是用户点击首页日程时触发的事件，该方法位于index.js的初始化page对象内部。页面在渲染日程时，会将日程的状态和key值渲染在节点上边，在用户点击日程时，这两个值会随事件参数传递到本方法中。这个方法只会处理状态为未开始和未结束的日程，未开始的日程点击后会通过actionSheet组件弹出修改按钮，未完成的日

程会通过actionSheet组件弹出修改和完成按钮，最后，不论用户点选了那种日程或者用户选择actionSheet中的那个选项，都是调用了相同的回调函数handleSelectTask方法操作日程，通过传递actionSheet的返回值和日程的状态来区分用户的操作。

代码清单9-10 点击日程事件

```
selectTask:function(e){
  //日程Key
  var key = e.currentTarget.dataset.key;
  //日程的当前状态
  var status = e.currentTarget.dataset.status;
  //未开始的日程：只展示修改按钮
  if(status === constant.taskStatus.pending){
    wx.showActionSheet({
      itemList:["修改"],
      complete:function(res){
        if(res.errMsg === 'showActionSheet:ok'){
          _fn.handleSelectTask(res.tapIndex, status, key);
        }
      }
    });
  }
  //已开始的日程：展示修改和完成按钮
} else if(status === constant.taskStatus.current){
  wx.showActionSheet({
    itemList:["修改","完成"],
    complete:function(res){
      if(res.errMsg === 'showActionSheet:ok'){
        _fn.handleSelectTask(res.tapIndex, status, key);
      }
    }
  });
}
}
```

代码清单9-11是操作日程的代码，在这段代码中一共涉及两个功能，一个是去日程详情页修改日程数据，另外一个是将用户选定的日程状态设置为完成。去日程详情页是通过方法_fn.goUpdateTask实现的，逻辑是拼接日程详情页的链接并设置参数pageType=update和key为用户选择的日程key。设置选定日程状态未完成的逻辑是，先从页面模

型的groupTask对象中读取未完成日程列表并从列表中得到用户选定的日程，将该日程的status设置为finish，将修改后的日程对象保存到storage中，最后调用_fn.init方法，同步storage中当天的日程数据。

代码清单9-11 操作日程

```
handleSelectTask:function(selectIdx,status,key){
    //未开始的日程
    if(status === constant.taskStatus.pending){
        if(selectIdx===0){//去日程详情页
            _fn.goUpdaeTask(key);
        }
    }
    //未结束的日程
    }else if(status === constant.taskStatus.current){
        if(selectIdx===0){//去日程详情页
            _fn.goUpdateTask(key);
        }else if(selectIdx===1){//修改日程状态为完成
            var taskList = _fn.getCurPage().data.groupTask.curList;
            //从groupTask对象中获取用户选定日程
            var task = taskList.filter(function(a){
                return a.key === key;
            })[0];
            task.status='finish';//修改状态
            //更新storage
            taskService.update({
                key:key,
                val:task
            });
            //重新读取当天的日程数据
            _fn.init();
        }
    }
},
```

9.3.3 日程管理页

日程管理页分为两个部分：月历部分和日程表部分。图9-8是日程管理页的数据模型，下面对其中的重要字段作解释：

- calendar: 与首页的calendar字段功能相似，用来渲染页头的日历组件。

- curDate: 当前页面默认选择的日期。

- grid: 用来渲染日程表格的数据对象。这个对象是一个二维数组。数组的横轴表示表格的一列，每一列的长度都是25，表示一天的24个小时。在这个数组中，有的元素为0，表示表格中的这个格子为空；有的元素是一个对象，表示表格中的这个格子是有日程的，需要渲染背景色，这个对象就是storage中的日程对象。二维数组的纵轴表示表格的列数。表格的列数是根据选定日期日程的开始结束时间综合计算得来的。

- timeLine: 用于生成24个小时的时间线数据。

```
▼ calendar: Array[7]
  ► 0: Object
  ► 1: Object
  ► 2: Object
  ► 3: Object
  ► 4: Object
  ► 5: Object
  ► 6: Object
  length: 7
  ► __proto__: Array[0]
  ► curDate: Moment
▼ grid: Array[4]
  ▼ 0: Array[25]
    0: 0
    1: 0
    2: 0
    3: 0
    4: 0
    5: 0
    6: 0
    7: 0
    8: 0
    9: 0
    10: 0
    11: 0
    12: 0
    13: 0
    14: 0
    15: 0
    16: 0
    17: 0
    18: 0
    19: 0
    20: 0
    21: 0
    ► 22: Object
    ► 23: Object
    24: 0
    length: 25
    ► __proto__: Array[0]
  ► 1: Array[25]
  ► 2: Array[25]
  ► 3: Array[25]
  length: 4
  ► __proto__: Array[0]
  ► timeLine: Array[25]
```

图9-8 页面数据模型

代码清单9-12是日程管理页加载的代码，与首页相同，本页面也调用了小程序的两个生命周期函数：`onLoad`和`onShow`方法。`onLoad`方法只会被调用一次，用来初始化日历和默认日期数据。`onShow`方法会在每次进入这个页面时被调用。日程详情页回来时同步`storage`中的选定日期的日程数据。

代码清单9-12 页面加载

```
onShow:function(){
  _fn.init();
},
onLoad:function(param){
  var ms = param.ms || new Date().getTime();
  _fn.getCurPage().data.curDate = moment(ms, 'x');
  _fn.getCalendar();
},
```

代码清单9-13是初始化日程表格数据的代码，在取得日程列表后，调用了`getTaskGrid`方法。该方法首先初始化了一个只有一个空列的表格数据。之后就是将日程列表中的日程数据添加到表格数据中，这里对每个日程都采用从左到右轮询日程表格的方式，检查当前日程是否可以添加到被轮询的列。如果可以则将其添加，不可以则继续轮询，如果最后还是不存在可以存放的列，则创建一个新列去存放。代码中有两个重要的变量：`process`和`lineIdx`，`process`用来标记是否继续循环，只有当程序为当前日程找到可以存放的列后，该值就会变为

false从而终止while循环。lineIdx是指每次while循环中，用于轮询列的编号。最后日程表格的数据格式就是一个二维数组。

代码清单9-13 生成日程表格

```
init:function(){  
    var moment = _fn.getCurPage().data.curDate || moment();  
    _fn.getTasks(moment,function(taskList){  
        _fn.getTaskGrid(taskList);  
    });  
}  
  
getTaskGrid:function(taskList){  
    //默认表格数据，只有一列全部为0  
    var grid = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0]];  
    for(var i = 0 ; i < taskList.length ; i++){  
        var task = taskList[i];  
        var process = true;  
        var lineIdx = 0;//当前遍历的列序号  
        while(process){  
            if(!grid[lineIdx]){//如果列不存在则创建  
                grid[lineIdx]=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0];  
            }  
            var line = grid[lineIdx];  
            //将日程添加到列中，如果成功结束结束循环，不成功则增加列序号，继续遍历下一列  
            var addRes = _fn.addTaskToLink(task,line);  
            if(addRes === true){  
                process = false;  
            }else{  
                lineIdx ++;  
            }  
        }  
    }  
    //将表格数据渲染到页面上  
    _fn.getCurPage().setData({  
        grid:grid, timeLine:[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,  
22,23,24]  
    });  
},
```

代码清单9-14是将日程添加到列的方法，方法的两个参数`task`和`line`分别是日程数据和表格的列数据。由于表格的每个格子都代表了一个小时，所以首先需要读取日程的开始时间和结束时间对应的小时数并判断开始时间和结束时间是否相同，如果相同，页面显示该日程不

会超过一个表格，否则会分布在多个连续的表格中，从而导致了两种不同的显示方案。第一种显示方案使用到了middlePer和startPos两个变量，middlePer是指日程在表格中的高度，startPos则是日程距离上边线的距离。第二种显示方案则用到了startPer和endPer，其分别代表了日程开始点距表格下边的高度和日程结束点距表格上边的高度，这些高度值是根据日程开始和结束时间的分钟值与60的比值得来的。在计算完这些样式所需的数据后，开始检查日程是否可以被添加到列中。列数据是一个长度为25的数组，数组元素的默认值为0，对应的渲染结果是一个空表格。检查日程是否可以被添加就是利用了这个特性，检查日程的开始小时到结束小时之间对应的列数据是否都为0，是则表示可以添加，否则则返回false，通知调用函数该列不可以将日程添加其中。当判断该列可以添加日程后，将日程数据，样式数据组成一个对象，添加到列数据中，并且返回true，通知调用函数成功将日程添加到列中。

代码清单9-14 日程添加到列

```
addTaskToLink:function(task,line){
    var startTime = moment(task.startTimeMs);
    var endTime = moment(task.endTimeMs);
    //日程开始的小时值
    var checkStart = startTime.hour()>=0?startTime.hour():0;
    //日程结束的小时值
    var checkEnd = endTime.hour()>=0?endTime.hour():0;
    //起点占表格的百分比
    var startPer;
    //终点占表格的百分比
    var endPer;
    //中间占表百分比
    var middlePer;
    var startPos;
    //开始点与结束点在同一个表格
    if(checkStart === checkEnd){
```

```

        startPos = (1-startTime.minute()/60)*100+'rpx';
        middlePer = ((endTime.minute() - startTime.minute())/60)*100+'%';
    }else{//开始点与结束点在不同的表格
        startPer = (1-startTime.minute()/60)*100+'%';
        endPer = (endTime.minute()/60)*100+'%';
    }

    //如果被检查列的开始到结束的元素中，又不是0的元素，则该列不能添加当前日程
    for(var i = checkStart ; i <= checkEnd ; i++){
        if(line[i]!==0){
            return false;
        }
    }
    //将表格的开始到结束元素全部幅值为日程数据
    for(var j = checkStart ; j <= checkEnd ; j++){
        var isMiddle = checkStart === checkEnd;
        var isStart,isEnd;
        if(!isMiddle){
            isStart = j===checkStart;
            isEnd = j===checkEnd;
        }
        line[j] = {
            middlePer:isMiddle?middlePer:null,
            startPos:isMiddle?startPos:null,
            startPer:isStart?startPer:null,
            endPer:isEnd?endPer:null,
            task:task
        };
    }
    return true;
}

```

代码清单9-15是渲染日程表格的代码。日程表格是一列一列的渲染完成，上文提到，列数据元素的默认值是0，渲染出来的是一个空的格子，如果列元素的值是一个日程对象，则向格子中添加一个有高度的view并用一种颜色将其填充。填充时会涉及几个问题：是否将表格填满？如果不满，上边缘留白多少？下边缘留白多少？表格不填满的情况一共有三种：第一种是日程的开始时间和结束时间在同一个小时内，即整个日程渲染在同一个格子内，日程的上下方都需要留白。这时渲染该格子的数据中会包含middlePer字段作为填充view的高度和startPos字段作为填充view的上边距。另外两种情况是在日程渲染多于一个格子的时候，日程的开始位置需要上方留白和日程的结束位

置下方需要留白。这时渲染该格子的数据中包含了startPer字段或者endPer字段，这两个字段都表示填充view的高度，但前者要求填充view靠底部展示，后者要求填充view靠顶部展示。

代码清单9-15 渲染日程表格

```
<view class="dayTaskWrapper">
  <block wx:for="{{grid}}" wx:for-item="gridItem" wx:for-index="gridIdx">
    <view class="task-column">
      <block wx:for="{{gridItem}}" wx:for-index="blockIdx">
        <view class="task-block taskTable-block" bindtap="chooseTime">
          <block wx:if="{{gridItem[blockIdx]!==0}}">
            <view data-taskKey="{{gridItem[blockIdx].task.key}}" class="task
            {{gridItem [blockIdx].startPer?'start':''}}
            {{gridItem[blockIdx].endPer?'end':''}} {{gridItem[blockIdx].task.important=== '一
            般'? 'imp- ortant_1': 'important_2'}} {{gridItem[blockIdx].task.status}}"
            style="height:
            {{gridItem[blockIdx].startPer||gridItem[blockIdx].endPer||gridItem[blockIdx].midd
            lePer}};margin-top:{{gridItem [blockIdx].startPos}}"></view>
          </block>
        </view>
      </block>
    </view>
  </block>
</view>
```

代码清单9-16是选择日期事件的代码。这段代码是在用户点击上方月历后触发，主要逻辑是高亮被点钟的日期，同是从storage中读取该日期用户保存的日程数据并重新渲染下方的日程表格。

代码清单9-16 选择日期事件

```
chooseDate:function(e){
  var ms = e.currentTarget.dataset.ms;
  _fn.getCurPage().data.curDate = moment(ms,'x');
  var calendar = _fn.getCurPage().data.calendar;
  //高亮被选中日期
  for(var i = 0 ; i < calendar.length ; i++){
    if(calendar[i].ms === ms*1){
      calendar[i].isSelect = true;
    }else{
      calendar[i].isSelect = false;
    }
  }
}
```

```
    }
    _fn.getCurPage().setData({calendar:calendar});
    //重新获取日程数据
    _fn.init();
}
```

代码清单9-17是选择日程的代码。用户在日程表格中点选日程后，会根据日程的状态在页面底部弹出浮层，浮层展示日程的基本详情以及修改或者完成的操作。

代码清单9-17 选择日程

```
chooseTask:function(e){
    var key = e.target.dataset.taskkey;
    if(!key){
        _fn.hideDetailPop();
    }else{
        if(_fn.getCurPage().data.selectTask &&
        _fn.getCurPage().data.selectTask.key===key){
            return;
        }
        taskService.get({key:key},function(res){
            res.data.curStatus = taskService.getStatus(res.data);
            _fn.getCurPage().setData({
                selectTask:res.data,
            });
            _fn.showDetailPop();
        });
    }
}
```

下面是跳转日程详情页的代码，在本页跳转日程详情页创建日程与首页跳转该页面创建日程多传递了一个需要创建日期的毫秒值。达到日程详情页可以根据用户需要创建不同日期的日程：

代码清单9-18 跳转日程详情页

```
addTask:function(e){
    var ms = _fn.getCurPage().data.curDate.valueOf();
    wx.navigateTo({
        url:'../create/create?pageType=create&ms='+ms
    });
}
```

```
}  });
```

9.4 小结

本章案例项目的难点有两个：

1) 设计和使用较为复杂的缓存：在日程表App的storage中保存日期数据和日程数据，这两个数据的关系是一对多。在代码结构设计时，将代码分成了数据操作层和业务层。数据操作层需要考虑向业务层提供同步和异步的方式读写数据的接口。业务层则只需专心搞好页面渲染和用户交互即可。

2) 将复杂的界面数据模型化：日程操作页的日程表格是一个渲染逻辑比较复杂的组件。业务的目的是形象地查看每个日程的时间长度。设计时就需要将这个业务抽象成为一个表格，之后思考表格中的每个格子怎么渲染，最后设计渲染数据，保证以最简单的方式达到业务的需求。

第10章 案例分析——多点商城

本章以多点App为原型，实现一套类似电商的小程序，多点App是一款网上超市购物软件，用户下单后可在2小时内送达。本案例忽略后台逻辑，主要实现多点App核心前端功能，其复杂度远远高于前几个案例，为了降低项目耦合度、提高项目可维护性、拓展性，我们结合小程序特性对项目进行了相应的架构优化，这些知识在前端项目中是通用的，大家在学习过程中除了细节技术，对前端项目架构一定要有清楚的认识，本章将会站在更高的高度剖析案例，构建逻辑代码中一些细节我们将适当忽略，学习过程中可以对比源码阅读，源码git：<https://github.com/wxapp-book/dmall.git>。本案例没使用ES6语法，运行源码时，不要勾选“开启ES6转ES5”选项，避免未知错误。

10.1 需求分析

通常电商网站按类型可以将页面分为2类，核心流程页面和辅助功能页面，如图10-1所示。



图10-1 电商页面类型

核心流程页面一般按顺序包含：入口页、商详页、购物车、结算页、支付页、支付状态页，这几个页面支撑了电商业务最核心的完整购买流程，无论什么形态的电商网站，这几个流程（页面）是必不可缺的，只是在表现形态上略有不同。页面流量从入口页到支付状态页呈漏斗状态，所有的电商公司都致力于提高漏斗转化率，提高支付成功的用户数量。在不同的电商App中，这些页面的表现形式也不同，如购物车页面，在大部分App中通常形态是一个单独的页面，而在个别App中购物车也可能是一个弹层。核心流程常见页面有：

- 入口页**：入口页一般是承接主流量、用户第一时间接触的页面，对性能、打开速度要求都比较高，电商App的首页、活动页都属于入口页，甚至有时在朋友圈中直接分享的商详页也属于入口页。入口页商品形态都比较丰富，经常变化，基于这些特性，入口页通常都是灵活可配的。

- 商详页**：商详页主要展示商品详情，也是访问量比较高的页面，商详页服务端通常会把一些常变的信息，如现价格、库存、促销、推荐等信息单独剥离出来，每次进入页面都请求，保证数据的及时性；而一些不常变的信息，如主图、商品名称、原价、参数规格、商品介绍等信息静态化，每次请求直接返回缓存中数据，当然，针对这些不常变的信息，前台也可以根据情况进行适当缓存，减少后端请求压力。

·**购物车**：购物车页面用于存放用户已添加的商品，在购物车中能对商品做简单的编辑，如修改数量，不同的电商形态对购物车商品有不同类型的划分，有些按店铺划分，有些按业务形态划分。

·**结算页**：结算页主要用于用户核实商品信息，填写收货地址、配送方式、收货时间等，除商品信息外和购买有关的其他信息，结算页一般不允许修改商品信息，因为在结算页中，后台会根据用户选择的商品配合促销规则、配送规则等实时计算出支付价格，如果结算商品不断变化会给后台造成很大压力，当然有些电商也在尝试将购物车和结算从交互上进行一定程度的融合，毕竟在电商网站中，少一次页面跳转就会缩短支付路径，提升用户转化率。

·**支付页**：支付页用于用户选择支付方式，它负责对接自己或第3方的支付系统，让用户以最便捷的方式完成支付。

·**支付状态页**：支付状态页告知用户当前支付结果，同时一些支付状态页也承担部分引流工作。

辅助功能页面 相对比较多且没有一个固定的模式，通常不同的电商公司根据自身提供的服务创建不同的辅助功能页面，而这些辅助功能页面提供的服务恰恰是每个电商App差异化所在，如一些O2O电商网站，地址是很核心的一个系统，进入App首先必须定位，然后根据定位信息获取周边服务，而一些B2C电商App对地址则不是太强化，只有在

下单时才会让用户填写地址，这两种方式各有优劣，总之，辅助功能页面和电商公司自身业务、形态息息相关，它们的存在是为了结合业务更好地让用户完成核心购买流程，提高漏斗转化率。辅助功能页面按类型划分有如下几种：

·[登录相关页面](#)：登录相关页面通常包含登录页、注册页、密码找回等功能性页面，它们负责用户整体登录态创建与维护，通常一个公司旗下产品会共用一套登录态，如腾讯、淘宝，甚至现在在不同平台也能共用登录态，如QQ、微信、微博的第三方平台登录，如果一个公司自己维护用户信息，前后端一定要做好安全方面的工作。

·[商品检索相关页面](#)：一般分类、搜索都属于商品检索页面，它能够将商品按用户更容易理解的纬度更直接的呈现给用户。

·[地址相关页面](#)：地址相关页面包含定位、地址列表、地址详情、地址修改等和地址信息维护相关页面，不同的公司有不同的组织方式。

·[订单相关页面](#)：订单相关页面包含，订单列表、订单详情等页面，在订单相关页面中，我们能了解当前订单的状态，同时可以对订单进行相应的操作，如取消、再次购买等，订单按类型、状态可被拆为多种形态，如按类型可划分为：**O2O**订单、**B2C**订单、海淘订单等，

按状态可划分为待支付、待收货、已完成（历史订单）等，不同的公司有不同的拆分方式。

·**个人中心相关页面：**个人中心页面包含用户信息、权益、会员特权等页面，不同用户所享有的权限、服务是不一样的，每个公司所提供的服务也不一样，所以个人中心根据业务不同有各种不同的形态，但整体上订单、地址修改、个人信息等入口都会放置在个人中心首页。

根据上述电商常见页面，结合多点App业务形态，本章实例实现了：首页、活动页、分类页、搜索页、购物车、结算页、支付页、支付状态页、个人中心页，根据交互要求，我们使用底部导航将首页、分类、购物车、用户中心整合为一级页面让用户能直接触达，同时活动页和首页都通过楼层搭建的方式实现，App部分界面如图10-2所示。



图10-2 多点商城界面

10.2 技术架构

在豆瓣电影实例中，我们为大家展示了通用的代码架构方式，本项目在代码架构上和豆瓣电影项目基本一致，具体细节我们就不再赘述，主要讲解一些架构上的差异与重点，本案例架构如图10-3所示。

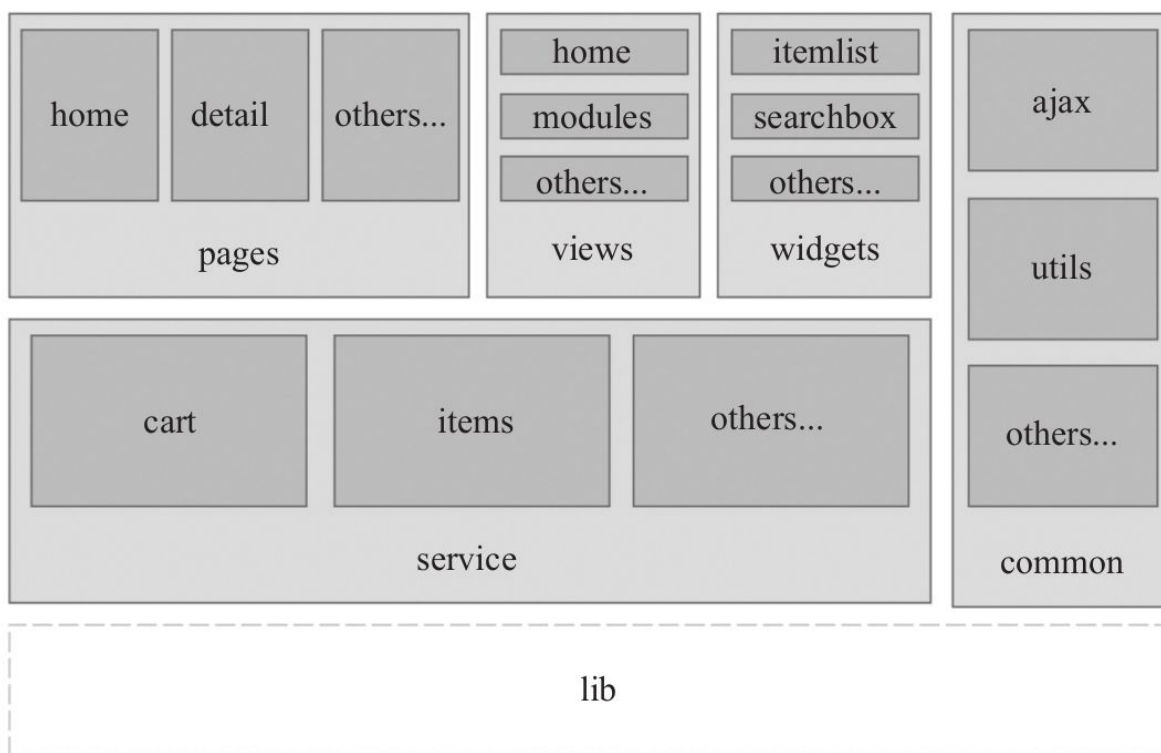


图10-3 多点商城架构

10.2.1 主界面架构

相比豆瓣电影，多点商城界面中多了一个**views**目录，这个目录用于存放主界面的4个视图，以及搭建活动时所需的楼层模板。多点商城主界面参见图10-4，通过底部导航实现跳转，点击底部导航分别跳首页、分类、购物车、个人中心，虽然小程序提供了**tabBar**配置，但是这个**tabBar**相对比较简单不能满足完整的业务需求，如小程序**tabBar**边框颜色只有白色和黑色，高度、大小不能控制，同时在交互中，购物车图标上需要显示目前购物车商品数量，这些都是**tabBar**无法实现的，所以我们决定实现一个自定义**tabBar**，并将相关页面以模板形式放置在**views**中进行管理。

我们自定义**tabBar**的思路是通过**fixed**布局，在底部固定一个<view/>，通过**template**实现不同的页面，点击不同bar时，调用相应的页面模板结合相应的业务数据进行渲染，在暴露数据接口时，我们定义了统一的数据结构格式，把业务数据封装在**data.currentData**中传递给视图层，这样保证调用模板的代码一致，主界面大致结构如图10-5所示。



1分钱
霸王购!

12.9 20:00正式开启
下单即抽iPhone 7 Plus

跨

早晚市

年

TOP榜

抢

满99减50

先

7折区

购

美通卡

- 每日推荐 -

优选爆品0点换新



Mixxtail魅夜 专业
鸡尾酒 蓝色幻...

¥9.00
¥8.00



曼可顿面包 香蕉
牛奶味 68g

¥2.25
¥1.65



康师傅蜂蜜绿茶
500ml

¥3.00
¥2.50



首页



分类



购物车

12



我的

图10-4 多点商城主界面

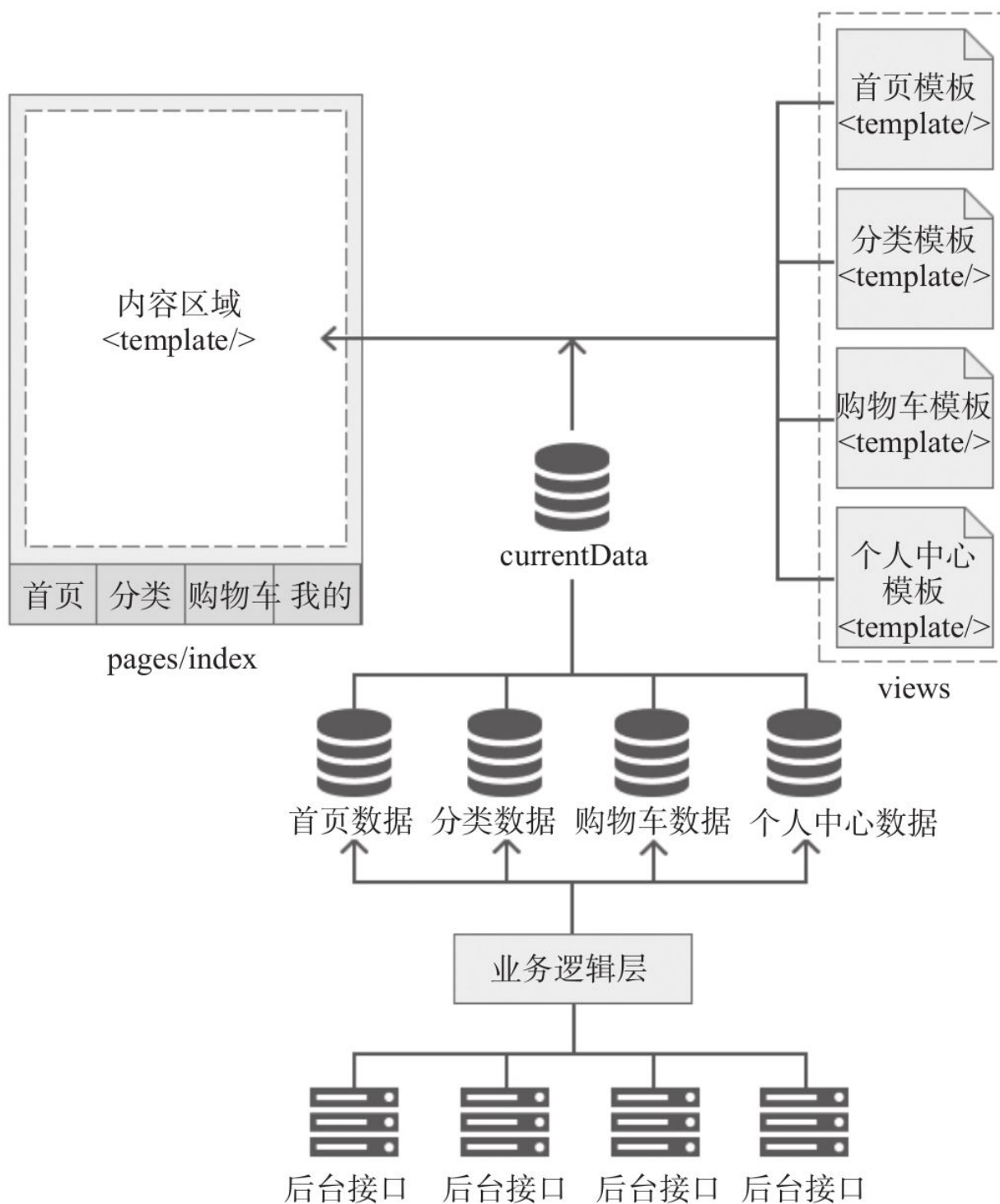


图10-5 主界面实现思路

主界面具体代码细节将在后文中进行分析。

10.2.2 业务逻辑层

在豆瓣电影中我们也创建了业务逻辑层，但实例业务比较简单，大家可能没能感觉到业务逻辑层的必要性，反而影响了代码可读性。而在多点商城中，由于不同的页面可能会调用同一个接口，这时接口层在解耦代码、减少代码量上都显得尤为重要。本例中我们创建了`cart`和`items`两个`services`，`cart`实现了`add`、`get`两个方法。`get`方法为主界面`tabBar`、商品详情提供购物车数量查询服务；`add`方法为首页、商品详情页、分类页提供添加购物车服务，`items`实现了`search`、`getDetail`两个方法，`search`为分类、搜索页提供商品查询服务，`getDetail`为商品详情页提供获取商品详情服务。在案例中，我们仅仅做了`services`层的应用展示，并没有完全实现相关接口和代码，业务逻辑层结构如图10-6所示。

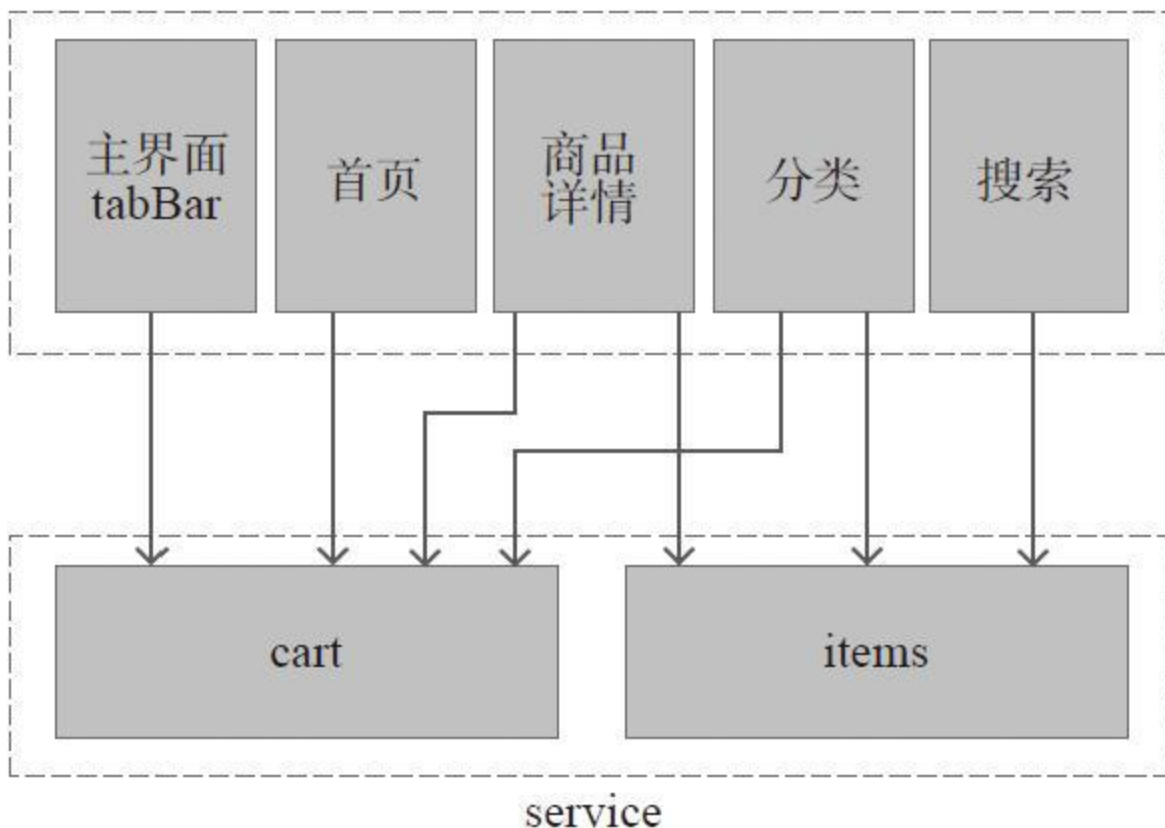


图10-6 业务逻辑层

在services/cart/cart.js，由于没有后台接口，我们直接在本地模拟购物车数量，代码如“代码清单10-1”。

代码清单10-1 业务逻辑层

```
var ajax = require( '../common/ajax/ajax.js' ),
    simularNumber = 12,
    handle;

handle = {
  // 添加购物车
  add : function( storeId, skuId, num, callback ) {
    ajax.query( {
      url : 'https://www.dmall.com',
      param : {
        storeId : storeId,
        skuId : skuId,
        num : num
      },
      callback : function() {
        .....
      }
    } );
  }
};
```

```

        // 模拟递增购物车数量
        ++simularNumber;
        callback( {
            errCode : '0000',
            errMsg : '',
            data : {
                num : simularNumber
            }
        } );
    }
}
},
// 获取购物数量
get : function( callback ) {
    ajax.query( {
        url : 'https://www.dmall.com',
        param : {},
        callback : function() {
            // 返回模拟数据
            callback( {
                errCode : '0000',
                errMsg : '',
                data : {
                    num : simularNumber
                }
            } );
        }
    } );
}
}
}
module.exports = handle;

```

在services/items/items.js中，search方法返回的数据是商品列表，这个数据的本地模拟数据在services/items/data.js中，getDetail则直接调用回调方法，由商详页面模拟返回数据，代码如下：

```

var ajax = require( '../common/ajax/ajax.js' ),
    data = require( 'data.js' ),
    handle;

handle = {
    search : function( param, callback ) {
        ajax.query( {
            url : 'https://www.dmall.com',
            param : param,
            callback : function( res ) {
                // 直接将返回数据透传给上层业务
                // 由上层业务根据错误码判断该进行如何交互
                callback( data );
            }
        } );
    },
    getDetail : function( param, callback ) {

```

```
    ajax.query( {  
      url : 'https://www.dmall.com',  
      param : param,  
      callback : function( res ) {  
        callback( res );  
      }  
    } );  
  }  
}  
  
module.exports = handle;
```

在业务逻辑层中我们除了提供业务接口，也可以对服务端返回数据做相应过滤，比如后台错误码通常是有含义的，我们可以跟进这些制定错误码，将返回数据过滤为上层业务所需的字段，如跳转链接、跳转方式、提示信息等，这是一种思路，具体实现没有统一规范，大家可根据项目实际情况选择合适的实现方式。

10.2.3 代理网络请求接口

客户端常常需要同后台进行交互，我们在小程序网络API基础上进行了再次封装，实现了一个网络请求的代理方法，主界面实现思路如图10-5所示，所有网络请求相关方法都封装在common/ajax/ajax.js文件中，这样做的原因有3点：

- 解耦底层代码。上层业务无需关心异步请求调用方式，底层可以随时根据情况统一修改请求实现方式。

- 针对自己项目格式化请求、返回参数，统一上游业务代码调用方式，当需要切换网络请求实现方式，或添加一些系统级别请求参数时，如用户信息、坐标信息等，我们可以在代理方法中实现。

- 对网络请求实现拦截。一些系统级别的错误码可以在这个代理层做统一的行为处理，如用户未登录、未授权、后台服务宕机等。

在前端项目中代理网络请求十分有必要，在封装请求参数时，我们也应该更多考虑整个系统的接口解耦，在过去网络端后台通常会直接读取cookie中某些字段判断用户登录态，而如果一个公司产品既有App又有H5，那么这样的接口在App中无法使用，所以我们建议将cookie或其他存储中的参数通过前端取值后赋值在请求参数中进行统一管理，这样能实现后台的解耦，无论前台技术怎样实现，有无cookie，

后台接口都能支持调用，而这类统一参数的封装，便可在代理网络请求接口中实现，实现原理如图10-7所示。

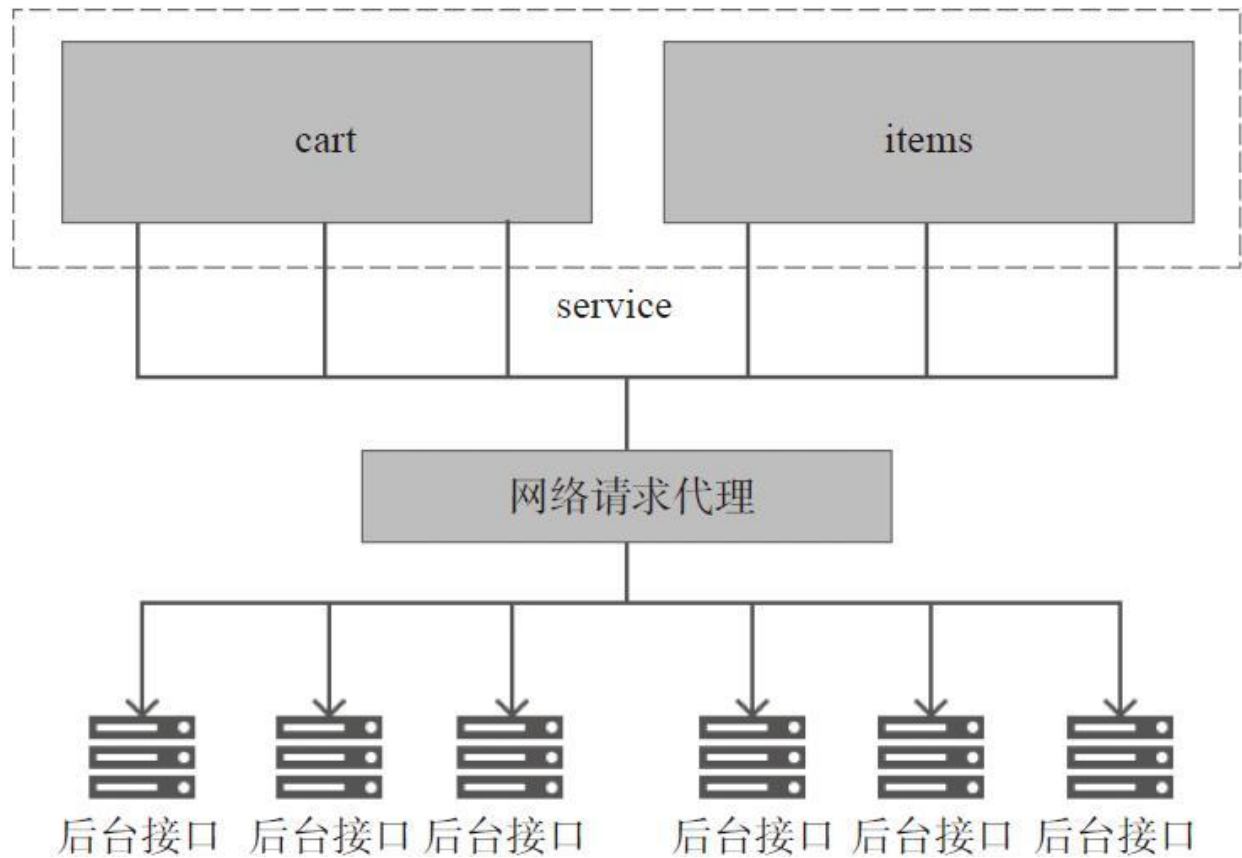


图10-7 代理网络请求接口

common/ajax/ajax.js文件代码如下：

```
var handle,
    _fn;

handle = {
    /* 统一请求接口 */
    query : function( object ) {
        wx.request({
            url : object.url,
            data : object.param,
            method : 'get',
            success : function( res ) {
                _fn.responseWrapper( res, object.callback );
            },
            fail : function( res ) {
```

```

        _fn.responseWrapper( res, object.callback );
    }
    });
}

_fn = {
    responseWrapper : function( res, callback ) {
        // 对请求失败，封装统一返回数据格式
        // 保证上层业务只传入一个回调方法，简化请求API
        if ( !res || res.statusCode !== 200 ) {
            callback( {
                errCode : -1,
                errMsg : '网络问题',
                data : {}
            } );
            return;
        }

        // 一些特殊登录统一拦截，如未登录等情况
        if ( res.data.errCode == 12341234 ) {
            // 跳转到登录页
            return;
        }
        if ( typeof callback == 'function' ) {
            // 正常回调，过滤小程序返回对象
            callback( res.data );
        }
    }
}

module.exports = handle;

```

业务层（services）调用请求代码如下，传入的回调方法只有一个：

```

handle = {
    search : function( param, callback ) {
        ajax.query( {
            url : 'https://www.dmall.com',
            param : param,
            callback : function( res ) {
                // 直接将返回数据透传给上层业务
                // 由上层业务根据错误码判断该进行如何交互
                callback( res );
            }
        } );
    }
}

```

10.2.4 本地模拟接口数据

自从Web项目推崇前后端分离以来，大大提高了项目整体开发效率，而小程序这种开发模式默认就是前后端分离的形式，针对这种前后端分离的项目，前端开发通常会在本地模拟一份接口数据格式。当前后两端各自调试完毕以后，直接修改services接口，去掉数据注入代码，便能直接对接后台接口，大大减少了前后端开发过程中不必要的沟通。在本案例中，我们基本将模拟数据data.js保存在页面目录中，如图10-8所示。这样我们在编写视图层和逻辑层代码时，可以随时根据需求修改数据格式，如pages/active/data.js，pages/cashier/data.js，views/cart/data.js等，当然也可以将这些模拟数据放置在不同service目录中，大家在开发前端项目过程中，一定要尽可能使用这种本地模拟数据的方式。

FOLDERS

▼ demo

▶ common

▶ pages

▶ service

▼ views

▼ cart

cart.js

cart.wxml

cart.wxss

data.js

▶ home

▶ mine

▶ modules

▶ sort

▶ widgets

app.js

app.json

app.wxss

图10-8 本地模拟接口数据

10.2.5 widgets

`widgets`与`common`层都是为项目提供一些公共的资源，不同之处在于，`widgets`更多偏向一个完整的UI组件块，比如下拉选择框、弹层框、搜索框等，`common`通常是一些公共的方法集合，没有相应的UI层，在本案例中，我们将分类页和搜索页的搜索框、商品列表抽象成`widgets`，如图10-9所示。

在小程序中由于数据模型、事件绑定都是在小程序页面文件中进行，我们虽然可以将`widgets`代码文件进行物理隔离，但真正对项目进行组件化有一定难度。在本案例中我们尝试了多种剥离方式，对于`widgets`的剥离，我们尝试仅仅剥离UI层，并没对逻辑层进行剥离，在列表页、搜索页引入模板渲染时，需要传入对应的事件方法名称，而具体的事件实现则是在列表页、搜索页自己的逻辑层中实现。而且有个细节需要注意，按照交互要求，在列表页中点击搜索框时需要跳转到搜索页，在搜索页`searchbox`失去焦点时需要根据用户输入搜索商品，所以我们对于这两个`wedgets`仅仅做了UI层的解耦，而实际项目中是否要采用这种解耦方式，还需要根据实际项目而定，本示例仅作参考。我们以`searchbox`为例，代码如下：



图10-9 widgets

```
<template name="search-box">
  <view class="search-box">
    <!-- 绑定调用页面传入的tapEvent方法名 -->
    <view class="box-cont" bindtap="{{tapEvent}}">
      <icon class="icon search"/>
      <view class="input" wx:if="{{!showInput}}">搜索多点超市商品</view>
      <!-- 绑定调用页面传入的blurEvent方法名 -->
      <input class="input" wx:else auto-focus="{{autoFocus}}" placeholder="搜索多点超市商品" bindblur="{{blurEvent}}"/>
    </view>
  </view>
</template>
```

在调用searchbox模板时，我们需要将对应的tapEvent、blurEvent事件通过参数传递给模板，达到同一模板在不同页面具备不同交互的效

果，这种方式更加依赖模板参数接口，每个调用页面都必须实现对应的方法，后面10.3.2节楼层搭建代码中，我们采用了另外一种方式，读者看完整个案例后可以对比两种实现方式的优劣。

10.2.6 全局样式控制

在项目样式管理中，我们将一些公共样式，如icon、默认轮播图、系统样式复写剥离到common/basestyle/basestyle.wxss文件中，在app.wxss文件引用basestyle.wxss文件，每个样式放置在.base-style类选择器下，如以下代码所示：

```
...
.base-style .icon.bar { background-image: url(http://static.dmall.com/kayak-project/wxdmall/dist/wxdmall/common/image/btm-bar-sprt.png); background-size: 500rpx auto; }

/* tabbar图标 */
.base-style .icon.bar.home { background-position: 0 0; }
.base-style .icon.bar.home.current { background-position: 0 -100rpx; }
.base-style .icon.bar.sort { background-position : -100rpx 0; }
.base-style .icon.bar.sort.current { background-position: -100rpx -100rpx; }
.base-style .icon.bar.cart { background-position: -300rpx 0; }
.base-style .icon.bar.cart.current { background-position: -300rpx -100rpx; }
.base-style .icon.bar.mine { background-position : -400rpx 0; }
.base-style .icon.bar.mine.current { background-position: -400rpx -100rpx; }
...
```

在使用时需要在构建代码外层添加类选择器，如下所示：

```
<view class="details base-style">
  <view class="slider">
    ...
  </view>

  <view class="info">
    ...
  </view>

  <view class="spec info-module">
    ...
  </view>

  <view class="more info-module">
    ...
  </view>

  <view class="toolbar">
    ...
  </view>
</view>
```

```
</view>  
</view>
```

采用这种方式能将公共的样式代码剥离到同一文件统一管理，同时在项目整体切换风格时，我们可以按类似方法创建另一个样式文件覆盖原有类选择器或新建类选择器，修改对应页面的class属性，或将页面base-style封装为数据注入，这样便可做到修改一处便整体切换页面风格。这种方式同时适用一些系统换肤的功能。

10.3 页面实现

上面讲解了多点商城项目主要架构和项目中的的一些核心难点，本小节讲解页面实现过程中一些核心技术和思路，如在首页与活动页中，我们使用了楼层搭建的方式动态创建页面；在其余一些页面中，我们尝试了多种解耦方式。这些编码思路没有绝对的好与不好，只有合适与不合适，在实际项目中一定要根据项目实际情况进行合理选择，避免过度架构。

10.3.1 主界面实现

主界面index中包含了一个tabBar和4个界面：首页、分类、购物车、个人中心，如图10-10所示。



多点 - 网上好超市



1分钱霸王购!

12.9 20:00正式开启
下单即抽iPhone 7 Plus

跨

早晚市

年

TOP榜

抢

满99减50

先

7折区

购

美通卡

- 每日推荐 -

优选爆品0点换新



Mixxtail魅夜 专业
鸡尾酒 蓝色幻...

¥9.00

¥8.00



曼可顿面包 香蕉
牛奶味 68g

¥2.25

¥1.65



康师傅蜂蜜绿茶
500ml

¥3.00

¥2.50



首页



分类



购物车

12



我的

图10-10 主界面

这4个页面整体架构思路在技术架构中进行了介绍，这里主要介绍主界面实现细节。主界面代码在pages/index目录中，首页、分类、购物车、个人中心界面分别单独剥离到views/home、views/sort、views/cart、views/mine目录中，在主界面文件中我们需要分别引入4个view对应的js、wxml、wxss文件，在每个view中必须实现render方法。在tab切换时会调用对应view的render方法。要注意的是，view目录中虽然有js、wxml、wxss三个文件，但它并不是小程序页面，仅仅是按模块化规则将代码进行了解耦，所以view中js是没有data对象的，也不能在页面逻辑中直接调用this.setData（）方法，所以在调用对应view的render方法时，我们将当前页面对象作为参数传递给对应view的render方法。代码片段见代码清单10-2。

代码清单10-2 主界面实现

```
...
// 注册当前页面对应js
views = {
  home : home,
  sort : sort,
  cart : cart,
  mine : mine
}

Page( {
  data : {...},
  onReady : function( ) {
    // 加载时默认选中首页
    _fn.selectView.call( this, 'home' );
  },
  onShow : function() {
    // 每次显示都刷新一次购物车
    // 这样保证在商详添加后在首页也能显示
    var self = this;
    serviceCart.get( function( res ) {
      self.setData( {
```

```

        'cart.num' : res.data.num
    } );
    } );
},
// 绑定到视图层切换tab页面方法
changeTab : function( e ) {
    var currentTarget = e.currentTarget,
        view = currentTarget.dataset.view;
    if ( !view ) { return; }
    // 调用对应的view
    _fn.selectView.call( this, view );
}
} );

_fn = {
    selectView : function( view ) {
        var view = views[view];
        if ( !view ) {
            return;
        }
        // 调用对应的render方法, 并将当前页面作为参数传递给对应模块
        view.render( this );
    }
}
}

```

在每个view中需要实现render方法，render方法必须实现两个逻辑：1）设置当前页面数据currentView和currentData，触发渲染；2）在当前页面对象中绑定当前view所需事件。这样每个view在研发过程中，可以单独在目录中创建自己的模拟数据、事件对象，在多人开发时不用都修改index.js文件，达到解耦目的。在事件绑定时，为了避免事件冲突，事件名称是view名+方法名，如sortChangeSort（分类view中切换分类tab选项）以分类为例，代码见代码清单10-3。

代码清单10-3 渲染的实现

```

var handle, events,...;

handle = {
    // callerPage为index Page对象
    render : function( callerPage ) {
        // 初始化界面
        _fn.init( callerPage );
        // 设置搜索框数据
        data.data.searchBox = {...}
        // 设置页面所需数据
        callerPage.setData( {

```

```

        currentView : 'sort',
        currentData : data.data
    } );
    // 选中第几个分类tab
    _fn.select( 0, callerPage );
}
};

// 需要绑定到index.js的事件方法
events = {
    sortChangeSort : function( e ) {...},
    sortGotoSearch : function( e ) {...},
    sortClickProxy : function( e ) {...}
}
// 私有方法
_fn = {
    init : function( callerPage ) {
        // 如果已经初始化则不需要再次触发，避免事件重复merge
        if ( callerPage.initedSort ) {
            return;
        }
        // 将事件绑定到index页面逻辑层
        utils.mix( callerPage, events );
        callerPage.initedSort = true;
    },
    // 选中对应分类
    select : function( index, callerPage ) {...},
    // 添加购物车
    addCart : function( e, callerPage ) {...}
}

module.exports = handle;

```

主界面渲染流程整体如图10-11所示。

在index模型中，currentData永远指向当前view所对应的数据模型，事件方法中，可以通过修改当前currentData而修改当前界面，在开发过程中可以在本地模拟sort数据模型，等前后台工作都开发完毕后，直接对接后台接口。

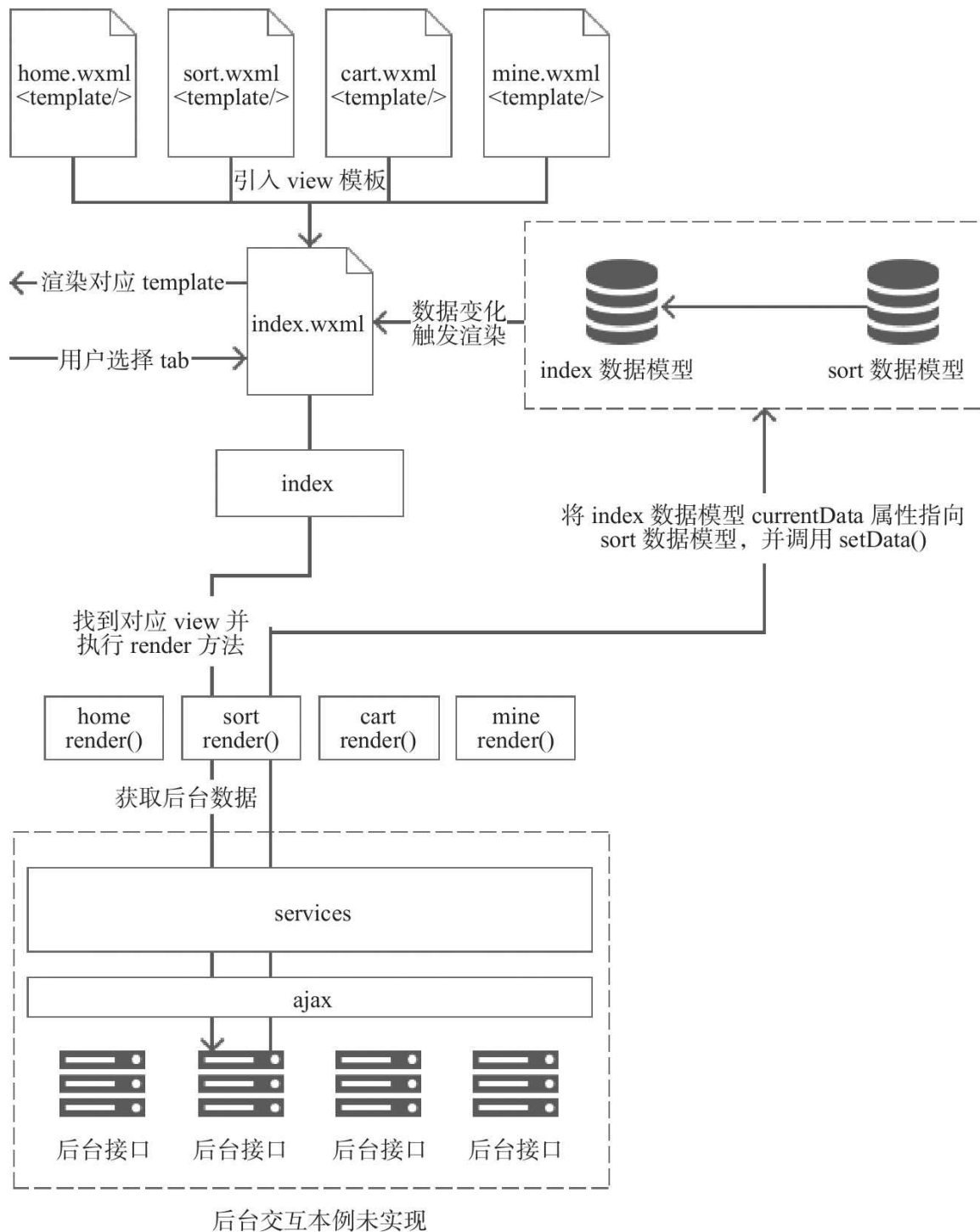


图10-11 主界面渲染流程

10.3.2 首页与活动页

首页与活动页的示例如图10-12所示。

首页与活动页由于常常变动，我们可以通过搭建楼层方式实现。在前端项目中这种动态搭建页面的技术已经十分成熟，本案例中楼层搭建通过模板实现。每一个楼层存放到一个模板中，渲染时，遍历数据模型列表，根据每个数据模型类型，找到对应模板进行渲染，本案例中首页和活动页都是采用这种楼层渲染方案，流程如图10-13所示。



图10-12 首页与活动页

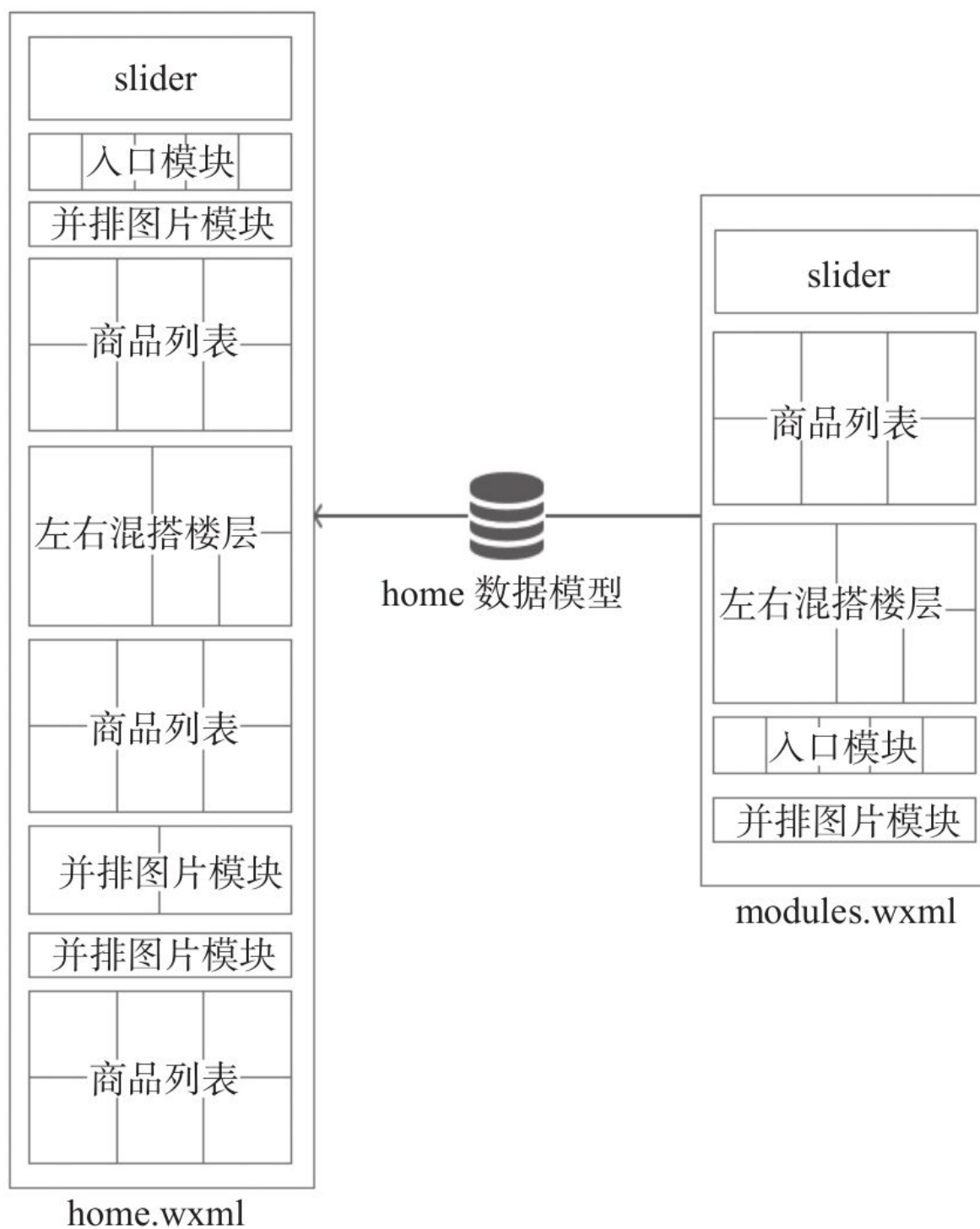


图10-13 首页渲染流程

所有楼层模板类型存放在views/modules/modules.wxml文件中，每个楼层对应的事件存放在views/modules/modules.js中，在渲染楼层页面时，除了引入楼层模板，还需要将modules.js中的事件合并到页面对象上。modules.wxml中我们定义了5类楼层：轮播图模块、入口模块、商品列表模块、左右混搭楼层模块、并排图片模块，代码整体结构如下：

```
<!-- 轮播图模块 -->
<template name="slider">
  <view class="slider">
    ...
  </view>
</template>

<!-- 入口模块 -->
<template name="entry">
  <view class="entry">
    ...
  </view>
</template>

<!-- 商品列表 -->
<template name="module-1">
  <view class="module-1 flex-col">
    ...
  </view>
</template>

<!-- 左右楼层混搭 -->
<template name="module-2">
  <view class="module-2 flex-col">
    ...
  </view>
</template>

<!-- 并排图片 -->
<template name="module-3">
  <view class="module-3 flex-col" style="height:
  {{data[0].height/data[0].width*750/data.length}}rpx;">
    ...
  </view>
</template>
```

以首页为例，modules数组中对应当前楼层的顺序，每个楼层包含两个属性type和data，type值为modules.wxml中对应楼层模板name值，

data为对应模板所需数据，页面数据模型如下：

```
data : {
  modules : [{
    // 轮播图模块
    type : 'slider',
    data : [...]
  }, {
    // 入口模块
    type : 'entry',
    data : [...]
  }, {
    // 并排图片模块
    type : 'module-3',
    data : [...]
  }, {
    // 商品列表模块
    type : 'module-1',
    data : [...]
  }, {
    // 左右混搭楼层模块
    type : 'module-3',
    data : [...]
  }, {
    // 并排图片模块
    type : 'module-3',
    data : [...]
  }, {
    // 商品列表模块
    type : 'module-1',
    data : [...]
  }
]}
```

根据这个模型，首页仅需遍历modules，调用对应模板进行渲染即可，代码如下：

```
<!-- 导入楼层模板 -->
<import src="../../modules/modules.wxml"/>

<template name="home">
  <view class="view-home modules">
    <!-- 遍历楼层 -->
    <block wx:for="{{modules}}">
      <!-- 调用对应模板 -->
      <template is="{{item.type}}" data="{{{...item, index}}}"></template>
    </block>
  </view>
</template>
```

活动页面实现和首页类似，唯一不同的是modules数据不一样。上文代码仅仅解决了楼层视图层渲染的问题，在实现过程中，每个楼层具备自己的事件，比如商品列表中有添加购物车事件，为了便于管理，我们将楼层本身的事件封装在modules.js中，调动楼层模板时，需要将楼层对应的方法合并到相应的页面对象中，如本案例modules.js已定义两个事件：

```
var utils = require( '../..common/utils/utils.js' ),
    serviceCart = require( '../..service/cart/cart.js' ),
    events, handle;

handle = {
  // 楼层对应事件
  events : {
    // 添加购物车
    modulesAddCart : function( e ) { ... },
    // 轮播图切换
    modulesSliderChange : function( e ) { ... }
  }
}

module.exports = handle;
```

调用页面时需要将事件添加到页面对象中：

```
...
// 引入modules
var modules = require( '../..views/modules/modules.js' ),
...

handle = {
  render : function( callerPage ) {
    _fn.init( callerPage );
    // 请求数据，渲染数据
    callerPage.setData( {
      currentView : 'home',
      currentData : data.data
    } );
  }
}

_fn = {
  init : function( callerPage ) {
    if ( callerPage.initedHome ) {
      return;
    }
    // 将楼层事件添加到页面中
    utils.mix( callerPage, modules.events );
  }
}
```

```
}  
}  
module.exports = handle;
```

楼层搭建的思路比较适合手机，由于屏幕尺寸和使用习惯关系，手机应用的界面都可以拆分为不同楼层，而对于一些大尺寸的设备（如PC、平板），动态搭建页面时需要在此引入栅格系统，首先将界面划分为不同的格子，然后将模块放置到对应的格子中，有兴趣的同学可以深入研究。

10.3.3 分类页与搜索页

分类页和搜索页的示例如图10-14所示。



图10-14 分类页和搜索页

分类页和搜索页在实现上本身没有什么难点，值得一提的是这两个页面中，我们尝试了另一种解耦方式。在之前的项目中我们都是把

事件方法放置在对应js文件中，调用页面时需要添加到当前页面对象中，而在分类和搜索中，因为同一组件在不同页面的行为不一样，如searchbox，在分类页面中按交互要求点击搜索框直接跳转到搜索页，在搜索页点击搜索框是输入搜索关键词，所以我们仅仅将页面视图进行解耦，制定每个组件对应的数据模型，每个组件数据模型中包含了组件相应行为控制和具体事件实现。以searchbox组件为例，组件模板中我们定义了tapEvent事件，代码如下：

```
<template name="search-box">
  <view class="search-box">
    <!-- 绑定用户传入的tapEvent方法名 -->
    <view class="box-cont" bindtap="{{tapEvent}}">
      <icon class="icon search"/>
      <view class="input" wx:if="{{!showInput}}">搜索多点超市商品</view>
      <!-- 绑定用户传入的blurEvent方法名 -->
      <input class="input" wx:else auto-focus="{{autoFocus}}" placeholder="搜索多点超市商品" bindblur="{{blurEvent}}"/>
    </view>
  </view>
</template>
```

在调用时将点击搜索框具体事件名通过tapEvent传入给searchbox，分类页面代码如下：

```
...
handle = {
  // callerPage为index Page对象
  render : function( callerPage ) {
    // 初始化界面
    _fn.init( callerPage );
    // 设置搜索框数据
    data.data.searchBox = {
      // 设置tapEvent对应事件名
      tapEvent : 'sortGotoSearch',
      // 不需要自动聚焦
      autoFocus : false,
      // 不显示输入框组件
      showInput : false
    }
    // 设置页面所需数据
    callerPage.setData( {
      currentView : 'sort',
      currentData : data.data
    })
  }
}
```

```
    } );
    ...
  }
};

// 需要绑定到index.js的事件方法
events = {
  // searchbox tapEvent具体方法实现
  sortChangeSort : function( e ) { ... },
  ...
}
// 私有方法
_fn = {
  init : function( callerPage ) {
    ...
    // 将事件绑定到index页面对象
    utils.mix( callerPage, events );
    ...
  },
  ...
}

module.exports = handle;
```

在搜索页中点击搜索框并没有任何交互，而是在进入搜索框时，searchbox需要默认聚焦，在searchbox失焦时需要搜索商品，所以searchBox数据对象实现代码如下：

```
Page( {
  data : {
    searchBox : {
      // 自动聚焦
      autoFocus : true,
      // 显示输入框组件
      showInput : true,
      // 失焦时触发搜索
      blurEvent : 'search'
    }
  },
  // 搜索方法具体实现
  search : function( e ) { ... }
} );
```

通过上述方式，我们能实现同一组件在不同页面实现不同交互的效果，这仅是一种解耦思路，而在实际项目中我们得反思是否有必要做这类解耦，在一些复杂度不高、代码量较小的页面，直接将组件构建代码写入页面反而效率更高。

10.3.4 支付流程

在通常项目中，支付流程涉及结算页、支付页、支付状态页，如图10-15所示。

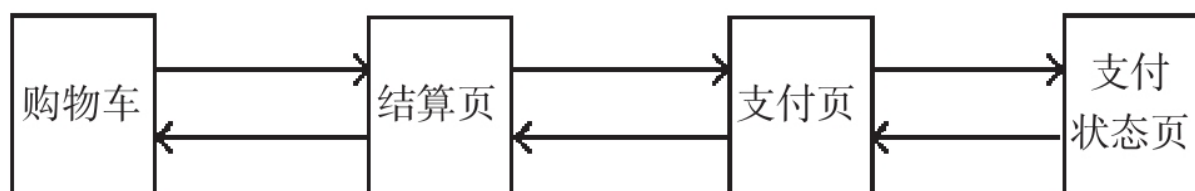


图10-15 普通支付流程

这几个页面需要做特殊的栈处理，通常这几个页面访问顺序为：用户首先访问结算页，然后访问支付页，最后跳转到支付状态页。但是在这种流程中如果用户在支付状态页点击返回按钮会导致页面返回到支付页，再返回到结算页，而此时支付页和结算页对应的订单已被处理，直接按这种流程返回会导致页面报错，甚至在H5项目中，接入支付宝之类的第3方支付，在支付页和支付状态页之间有更长的支付流程，这时用户点击“返回”按钮时会返回到之前访问过的页面。同时在交互体验上，当支付成功后点击“返回”按钮，应该返回至首页或结算页上一级页面，所以我们对此流程进行了特殊处理，如图10-16所示。

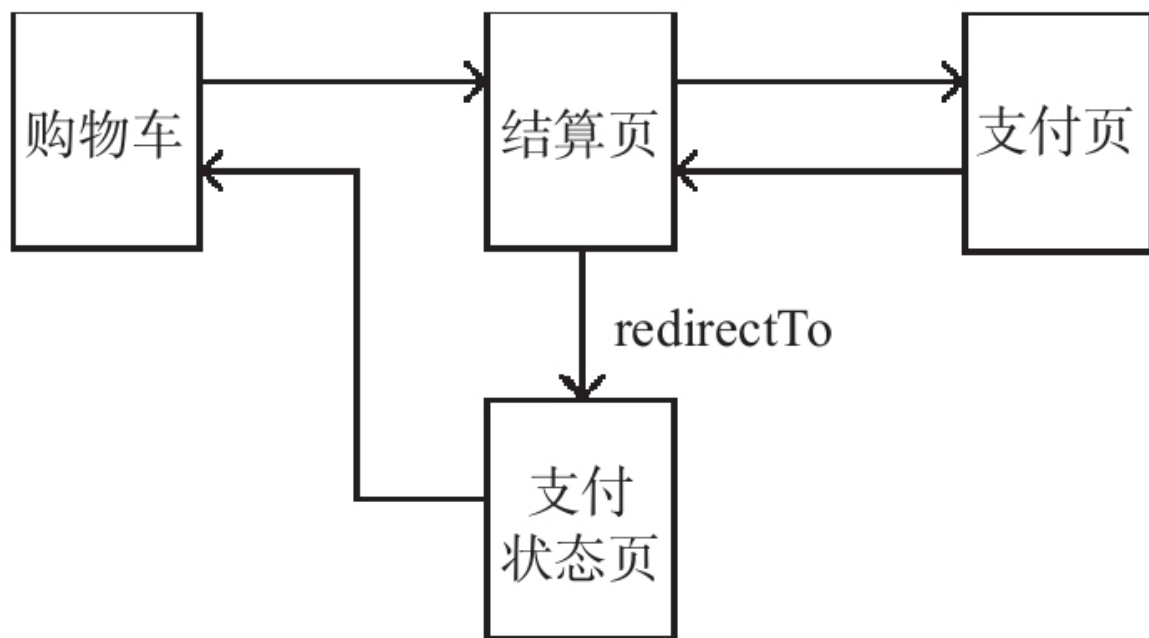


图10-16 优化后支付流程

在支付页支付成功后直接返回到结算页，结算页根据之前生成的订单，将当前页面replace到支付状态页，支付状态页根据支付订单查询支付结果并显示，这样，当用户在状态页点击“返回”按钮时，便能直接返回购物车。在H5项目中，可以在结算页跳转前调用 `history.replaceState()` 方法将当前栈直接修改为支付状态页，这样支付页支付成功后就能直接跳转到支付状态页，同时在支付页时可将当前栈位置存储在 `sessionStorage` 中，在第三方支付成功回调地址中判断当前栈与之前支付页栈的步长，调用 `history.back(step+1)`；跳转到结算页，这样就能避免用户点击返回时出现之前已访问过的页面。

10.3.5 其他页面

其他页面在实现上大同小异，在实现过程中有几个细节需要注意：

- 设计页面前一定要认真思考页面数据模型，好的数据模型能大大减少页面工作量。

- 在小程序中不能像浏览器一样直接获取组件尺寸，在商详页或一些需要展示图片的页面中，可以让后台返回图片尺寸或调用小程序API获取图片尺寸，渲染时根据屏幕尺寸进行等比缩放。

- 在客户端中小程序页面整体放置在一个view容器中，当前页面如果高度不够时背景色为白色，这个容器底色是不能通过设置json文件中window背景色改变的，所以在编写页面时，可以将当前页面最小高度设置为屏幕高度，这样能让页面完全填充界面。

- 编写样式文件时，尽量使用类选择器隔离当前样式，避免全局样式污染。

10.4 小结

本章案例基本覆盖小程序前端常用技术，可以直接沿用到任何项目中。在项目过程中我们尝试了多种解耦方式，而解耦的目的是为了降低项目风险，便于多人开发，但在编写多点商城案例过程中，我们也在反思小程序的本质是什么？像商城这类平台类型的产品是否适合做成小程序？小程序是否适合一些轻量级的应用？如果小程序都是一些轻量级、功能性的页面，那项目的整体架构是否可以适当简化？总之，在小程序的道路上我们还有很多路要走，大家一起加油！